thoughtworks

Design. Engineering. AI.

技术雷达

针对当今科技领域发展的
前沿指南

第33期
2025年11月

关于技术雷达

Thoughtworkers 始终对技术怀有炽热激情。我们致力于构建技术、开展研究、实施测试、推动开源、撰写洞见，并不懈追求技术的持续优化——这一切努力皆以普惠大众为宗旨。我们的使命是倡导软件卓越理念，引领信息技术革命。为此，我们创建并持续分享 Thoughtworks 技术雷达，将其作为实现这一使命的重要载体。

Thoughtworks 技术顾问委员会——由公司资深技术领袖组成的核心团队——负责技术雷达的编撰工作。该委员会定期召开会议，深入探讨 Thoughtworks 全球技术战略，以及对本行业产生深远影响的技术趋势。技术雷达以高度凝练的形式，系统呈现技术顾问委员会研讨成果，旨在为从开发者至首席技术官等广泛利益相关方提供价值参考。

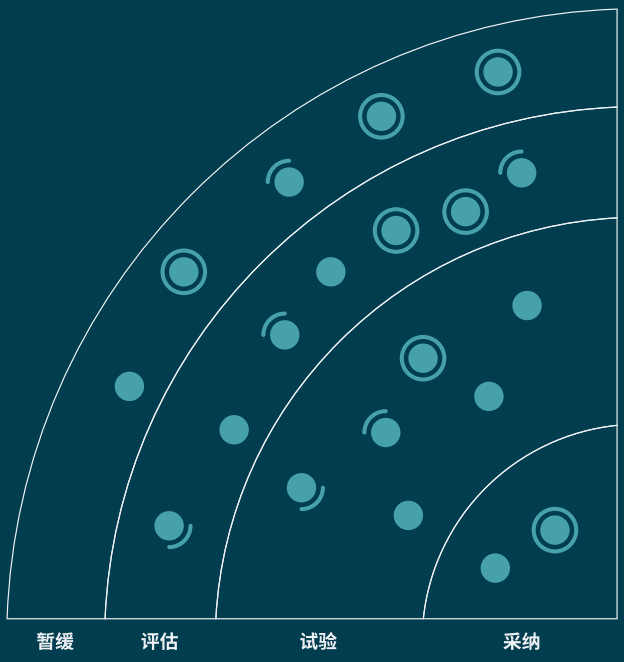
我们诚挚邀请您共同探索这些前沿技术。技术雷达采用可视化设计，将技术要素划分为四大象限：技术、工具、平台以及语言与框架。对于可能跨象限分布的技术条目，我们依据其核心属性进行精准归类。同时，通过四个同心环的布局设计，清晰展现我们对各项技术当前所处发展阶段的专业判断。

如需获取技术雷达的更多背景信息，请访问 thoughtworks.com/cn/radar/faq

雷达一览

技术雷达的核心使命是追踪具有价值的技术动向，我们将其定义为技术脉冲。本雷达通过两大核心维度对技术脉冲进行系统化组织：象限与评估环。技术象限区分技术脉冲的领域属性，评估环则体现我们对各项技术的应用建议层级。

技术脉冲指在软件开发领域产生影响力的技术或方法。这些脉冲始终处于动态演进之中——其在雷达图谱中的位置会持续变化——通常随着评估环的递进，标志着我们对其推荐力度的逐步增强。



采纳：我们坚定认为行业应当广泛采用此类技术。在项目条件适配时，我们会优先应用这些技术。

试验：具备显著探索价值。建议重点构建相关技术能力，企业可在风险可控的项目中开展实践验证。

评估：值得进行深度调研，重点评估其对组织产生的潜在影响。

暂缓：建议采取审慎态度对待技术应用。

新的　挪进 / 挪出　没有变化

技术雷达具有前瞻导向性。为保持内容时效性，我们对近期未发生位移的技术条目进行视觉淡化处理，此举并非否定其技术价值，而是受限于雷达版面的空间约束。

贡献者

技术顾问委员会（Technology Advisory Board，简称 TAB）由 22 位资深技术专家组成，是 Thoughtworks 技术治理体系的核心智库。该委员会采用 " 线下深度研讨 + 线上持续协同 " 的运作机制：每年举行两次线下全体会议，双周召开线上例会。其核心职能是为 Thoughtworks 首席技术官 Rachel Laycock 提供战略决策支持，驱动企业技术愿景的落地实施。

作为跨领域技术治理机构，TAB 聚焦影响企业技术演进与技术人才发展的关键议题，构建覆盖技术战略、创新实践与组织能力的全景洞察。本年度技术雷达的内容体系，源于该委员会 2025 年 9 月在布加勒斯特举行的专项研讨成果，汇集了全球顶尖技术专家对行业趋势的前瞻判断。


Rachel Laycock
首席技术官


Martin Fowler
首席科学家

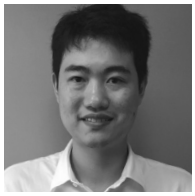

Alessio Ferri


Bharani Subramaniam


Birgitta Böckeler


Bryan Oliver


Camilla Falconi Crispim


Chris Chakrit Riddhagni


Effy Elden


James Lewis


Kief Morris


Ken Mugrage


Maya Ormaza


Nati Rivera


Neal Ford


Ni Wang
王妮


Nimisha Asthagiri


Pawan Shah


Selvakumar Natesan


Shangqi Liu
刘尚奇


Vanya Seth


Will Amaral

制作团队人员

编审委员会

- **William Amaral**
  产品负责人
- **Nati Rivera**
  产品负责人
- **Preeti Mishra**
  项目及活动经理
- **Richard Gall**
  内容编辑
- **Michael Koch**
  文字编辑
- **Gareth Morgan**
  内容与思想领导力总监

设计与多媒体部

- **Leticia Nunes**
  首席设计师
- **Sruba Deb**
  视觉设计师
- **Ryan Cambage**
  多媒体专家
- **Anish Thomas**
  多媒体设计师
- **唐蓉**
  中文版设计师

数字与网络体验部

- **Rashmi Naganur**
  业务分析师
- **Brigitte Britten-Kelly**
  数字内容策略师
- **Vandita Kamboj**
  用户体验设计师
- **Lohith Amruthappa**
  数据分析专家
- **Neeti Thakur**
  营销自动化专员

传播与公共关系

- **Shalini Jagadish**
  内部传播专员
- **Hiral Shah**
  社交媒体运营专家
- **Abhishek Kasegaonkar**
  社交媒体运营专家
- **Michelle Surendran**
  公共关系专员
- **Anushree Tapuriah**
  活动与广告策划专家
- **Prakhar Nigam**
  活动与广告策划专家

中文翻译

- **张霄翀**
  专家级软件开发工程师
- **廖燊**
  软件开发工程师
- **冯炜**
  高级DevOps工程师

本期主题

基础设施编排助力 AI 发展

AI 工作负载正在推动各组织对大规模 GPU 阵列进行编排，以支持训练和推理。团队处理的模型规模日益超出单个加速器的容量（即使配备 80GB HBM），这促使他们转向分布式训练和多 GPU 推理。因此，平台团队正在构建复杂的多阶段流水线，并持续调优吞吐量和延迟。在该领域的讨论包括用于集群遥测的 Nvidia DCGM Exporter，以及将作业部署在互连带宽最高位置的 拓扑感知调度。

在 GPU 需求激增之前，Kubernetes 已经是容器编排的事实标准——即便我们也探索了 micro 和 Uncloud 等替代方案，它仍是大规模管理 AI 工作负载的坚实基础。我们正在跟踪新兴的 GPU 感知调度模式——如通过 Kueue 实现队列和配额管理，并结合拓扑感知放置和团体调度（gang scheduling）——以便将多 GPU 作业部署在高速 GPU 间链路（例如 NVLink/NVSwitch）和连续的数据中心“岛屿”（例如配备 RDMA 的机架或 pod）内。Kubernetes 最近的多 GPU 和 NUMA 感知 API 改进进一步增强了这些能力，提高了跨设备带宽，降低了尾延迟，并提升了有效利用率。

我们预计，随着平台团队竞相支持日益增长的人工智能编码工作流程需求以及 MCP 推动的智能体崛起，人工智能基础设施将迎来快速创新。我们认为，GPU 感知编排正成为基本要求——拓扑结构如今已成为首要的调度考量因素。

MCP 推动的智能体崛起

MCP 与智能体的双重崛起——以及围绕它们构建的不断扩展的协议和工具生态——主导了本期技术雷达。几乎所有主要厂商都在为其工具增加 MCP 支持，这是有道理的：在许多方面，MCP 已成为推动智能体运行并使其高效半自主工作的终极集成协议。这些能力对于提高智能体工作流的生产力至关重要。

我们观察到智能体工作流的持续创新，其中 上下文工程 已被证明对于优化行为和资源消耗至关重要。新协议如 A2A 和 AG-UI 正在减少构建和扩展面向用户的多智能体应用所需的模板代码。在软件开发领域，我们比较了向编码智能体提供上下文的不同方式——从 AGENTS.md 文件到像 1 的模式。正如 AI 生态系统中常见的，每期技术雷达都带来一波新的创新——上次是 RAG；这次是智能体工作流及支持它们的日益丰富的工具、技术和平台，同时还有一些值得关注的新兴 AI 反模式。

AI 编码工作流

显而易见，AI 正在重塑我们构建和维护软件的方式，并持续主导着我们近期的讨论。随着 AI 在整个软件价值链中战略性地嵌入——从 使用 AI 理解遗留代码库 到 面向前向工程的生成式 AI——我们正在学习如何更好地为编码智能体提供知识支持。各团队正在尝试新的实践，例如通过 AGENTS.md 文件定义自定义指令，以及集成 Context7 等 MCP 服务器以获取最新的依赖文档。

同时，业界越来越意识到，AI 必须赋能整个团队，而不仅仅是个人贡献者。像共享指令库 和 自定义命令等技术正逐渐兴起，以确保知识的公平扩散。工具生态也日趋活跃：设计师们正在探索 UX Pilot 和 AI Design Reviewer，而开发者们则借助 v0 和 Bolt 快速进行自助式 UI 原型设计。

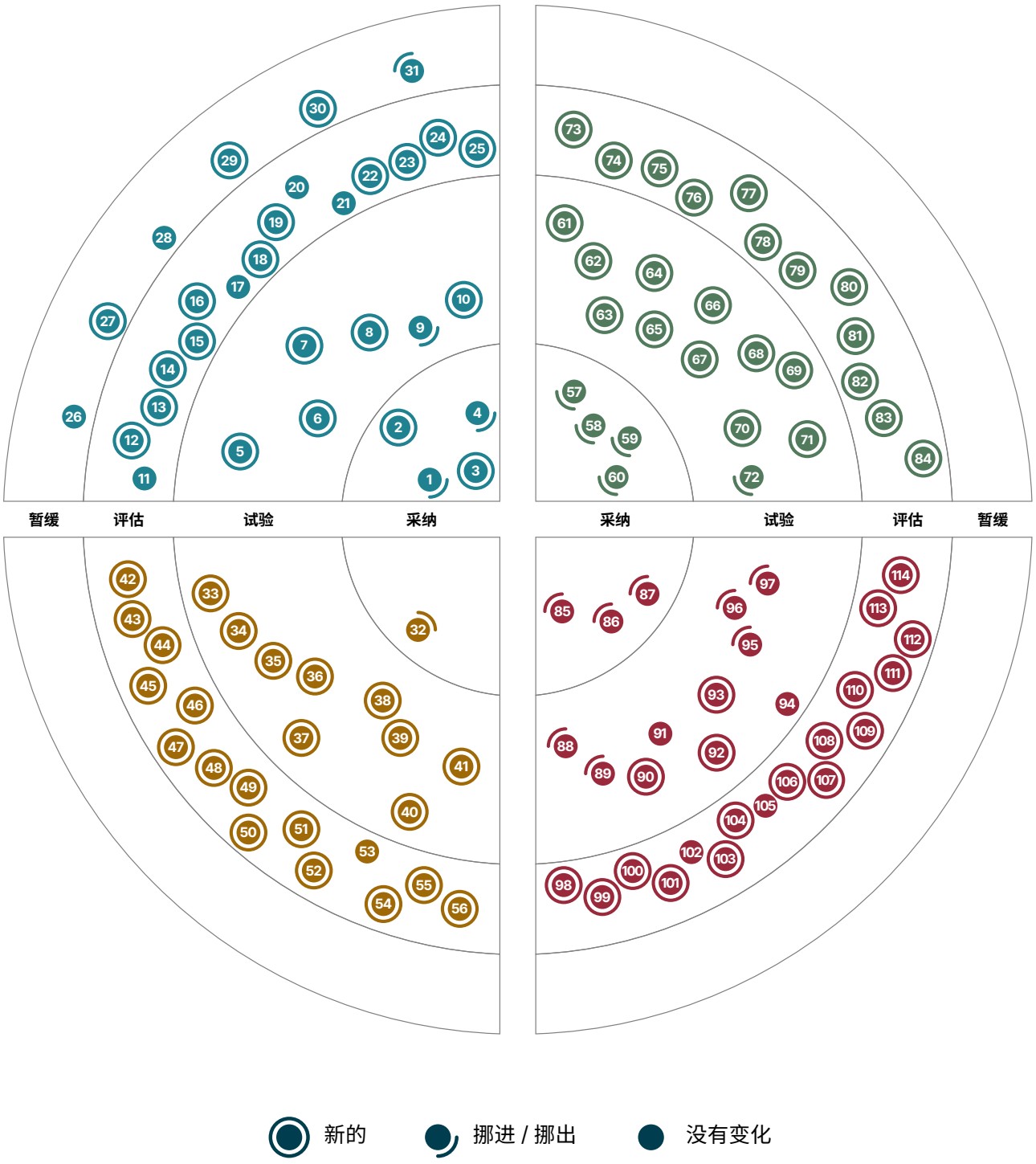
我们也在持续讨论规格驱动开发——包括其适用范围、粒度，以及作为增量交付单一事实源的潜力。然而，在这股 AI 热潮中，对 AI 生成代码的盲目信任依然是共同的担忧，这提醒我们：尽管 AI 能加速工程进程，但人类的判断力仍然不可或缺。

新兴 AI 反模式

AI 在各行业的加速应用，既催生了有效的实践，也涌现出一些反模式。我们看到类似 用 GenAI 自助、进行一次性 UI 原型设计 的概念确实具有明显的实用性，但也意识到它们可能将组织引向 AI 加速影子 IT 这种反模式。同样，随着 模型上下文协议（MCP）的普及，许多团队也容易陷入 天真的 API 到 MCP 转换 这种反模式。

我们还发现 Text to SQL 解决方案的实际效果并未达到最初的预期，而 对 AI 生成代码的自满情绪 依然是一个值得关注的问题。即便是在诸如 规范驱动开发 等新兴实践中，我们也注意到有回归传统软件工程反模式的风险——最明显的是，过度依赖前期规范制定和一次性大规模发布。由于生成式 AI 正以前所未有的速度和规模发展，我们预计新的反模式将迅速涌现。团队应保持警惕，关注那些乍看有效但随着时间推移而失效、反馈迟缓、适应性差或责任不清的模式。

本期雷达



本期雷达

技术

采纳

1. 持续合规
2. 为软件团队精选共享指令
3. Pre-commit hooks
4. 使用生成式 AI 来理解遗留代码库

试验

5. AGENTS.md
6. AI 用于代码迁移
7. Delta Lake liquid clustering
8. 使用 GenAI 的自助式 UI 原型设计
9. 从 LLMs 获取结构化输出
10. 测试 && 提交 || 回退 (Test && Commit || Revert)

评估

11. AI 驱动的 UI 测试
12. 将编码智能体锚定到参考应用
13. 上下文工程
14. 用于前向工程的生成式 AI
15. 将 GraphQL 作为 LLM 的数据访问模式
16. 知识流量胜于知识存量
17. 将 LLM 用作评审
18. 设备端信息检索
19. SAIF
20. 无边车服务网格
21. 小语言模型 (SLMs)
22. 规范驱动开发（Spec-driven development）
23. 编码智能体团队
24. 拓扑感知调度
25. AI 有害流程分析

暂缓

26. AI 加速影子 IT（AI-accelerated Shadow IT）
27. 容量驱动开发
28. 自满于 AI 生成的代码
29. 天真的 API 到 MCP 转换
30. 独立数据工程团队
31. Text to SQL

平台

采纳

32. 云上 Arm

试验

33. Apache Paimon
34. DataDog LLM Observability
35. Delta Sharing
36. Dovetail
37. Langdock
38. LangSmith
39. 模型上下文协议（MCP）
40. n8n
41. OpenThread

评估

42. AG-UI 协议
43. Agent-to-Agent (A2A) 协议
44. Amazon S3 Vectors
45. Ardoq
46. CloudNativePG
47. Coder
48. Graft
49. groundcover
50. Karmada
51. OpenFeature
52. Oxide
53. Restate
54. SkyPilot
55. StarRocks
56. Uncloud

暂缓

—

本期雷达

工具

采纳

57. ClickHouse
58. NeMo Guardrails
59. pnpm
60. Pydantic

试验

61. AI Design Reviewer
62. Barman
63. Claude Code
64. Cleanlab
65. Context7
66. 数据契约 CLI
67. Databricks Assistant
68. Hoppscotch
69. NVIDIA DCGM Exporter
70. RelationalAI
71. UX Pilot
72. v0

评估

73. Augment Code
74. Azure AI 文档智能
75. Docling
76. E2B
77. Helix 编辑器
78. Kueue
79. MCP-Scan
80. oRPC
81. Power user for dbt
82. Serena
83. SweetPad
84. Tape/Z（用于 Z/OS 的汇编程序分析工具集）

暂缓

—

语言和框架

采纳

85. Fastify
86. LangGraph
87. vLLM

试验

88. Crossplane
89. DeepEval
90. FastMCP
91. LiteLLM
92. MLForecast
93. Nuxt
94. Phoenix
95. Presidio
96. Pydantic AI
97. Tauri

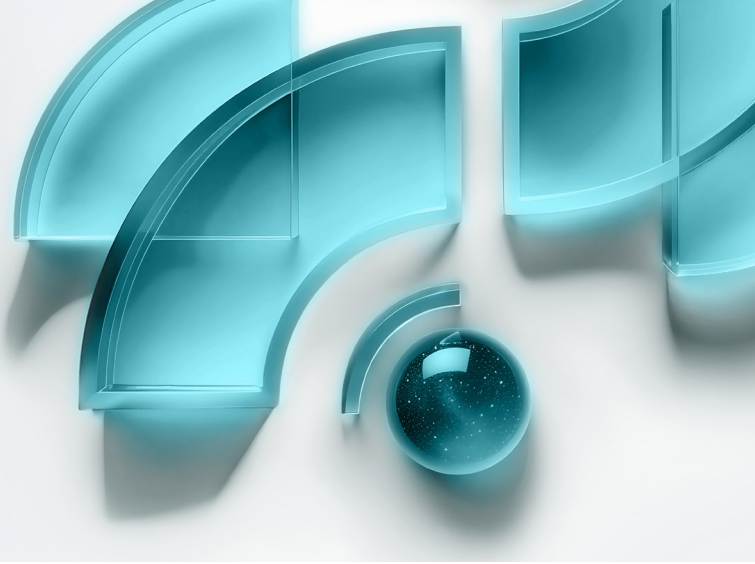
评估

98. Agent Development Kit（ADK）
99. Agno
100. assistant-ui
101. AutoRound
102. Browser Use
103. DeepSpeed
104. Drizzle
105. Java 后量子密码学
106. kagent
107. LangExtract
108. Langflow
109. LMCache
110. Mem0
111. 开放安全控制评估语言 (OSCAL)
112. OpenInference
113. Valibot
114. Vercel AI SDK

暂缓

—

技术



采纳

1. 持续合规
2. 为软件团队精选共享指令
3. Pre-commit hooks
4. 使用生成式 AI 来理解遗留代码库

试验

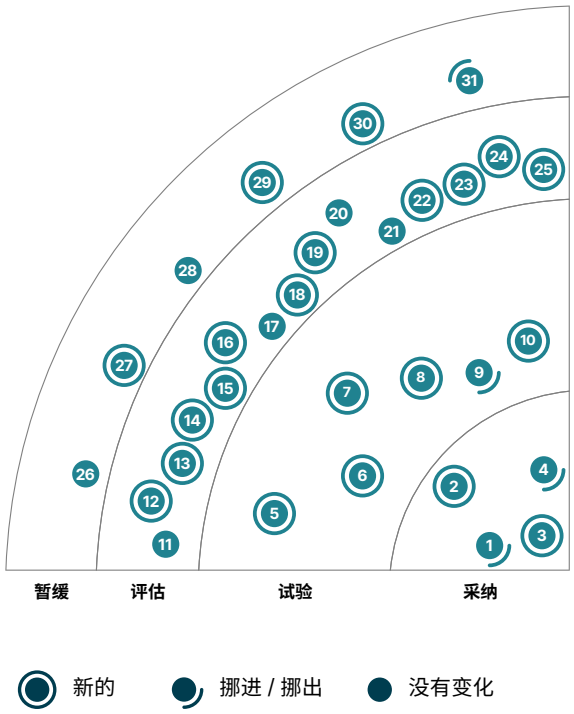
5. AGENTS.md
6. AI 用于代码迁移
7. Delta Lake liquid clustering
8. 使用 GenAI 的自助式 UI 原型设计
9. 从 LLMs 获取结构化输出
10. 测试 && 提交 || 回退 (Test && Commit || Revert)

评估

11. AI 驱动的 UI 测试
12. 将编码智能体锚定到参考应用
13. 上下文工程
14. 用于前向工程的生成式 AI
15. 将 GraphQL 作为 LLM 的数据访问模式
16. 知识流量胜于知识存量
17. 将 LLM 用作评审
18. 设备端信息检索
19. SAIF
20. 无边车服务网格
21. 小语言模型 (SLMs)
22. 规范驱动开发（Spec-driven development）
23. 编码智能体团队
24. 拓扑感知调度
25. AI 有害流程分析

暂缓

26. AI 加速影子 IT（AI-accelerated Shadow IT）
27. 容量驱动开发
28. 自满于 AI 生成的代码
29. 天真的 API 到 MCP 转换
30. 独立数据工程团队
31. Text to SQL



1. 持续合规

采纳

持续合规 是通过自动化确保软件开发过程和技术持续满足监管和安全标准的实践。手动合规检查可能会减缓开发速度并引入人为错误，而自动化检查和审计提供更快的反馈、更清晰的证据和简化的报告。

通过集成安全策略即代码工具（如 Open Policy Agent）并在 CD 管道中生成 SBOM——与 SLSA 指导保持一致——团队可以及早发现和解决合规问题。将规则和最佳实践编码化可以在不造成瓶颈的情况下，在团队间一致地执行标准。开放安全控制评估语言 (OSCAL) 也显示出作为大规模自动化合规框架的前景。

持续合规的实践和工具现在已经足够成熟，应该被视为合理的默认选择，因此我们将推荐级别提升至“采纳”。AI 在编码中日益增多的使用——以及随之而来的对 AI 生成代码的自满风险，使得将合规性融入开发流程比以往任何时候都更加重要。

2. 为软件团队精选共享指令

采纳

对于积极在软件交付中使用 AI 的团队，下一步是超越个人提示词，转向为软件团队精选共享指令。此做法通过分享经过验证的高质量指令，让你能够将 AI 高效地运用于所有交付任务——而不仅仅是编码。最直接的实现方式，是像提交 AGENTS.md 之类的指令文件一样，将它们直接纳入项目仓库。大多数 AI 编码工具——包括 Cursor、Windsurf 和 Claude Code——都支持通过自定义斜杠命令或工作流共享指令。对于非编码任务，你可以建立开箱即用的组织级提示词库。这种系统化方法便于持续改进：每当有提示词被优化，全体成员都可受益，从而确保大家始终使用最佳的 AI 指令。

3. Pre-commit hooks

采纳

Git hooks 已经存在很长一段时间了，但我们认为它们仍然被低估和未充分使用。随着 AI 辅助编码和自主智能编程的兴起，意外提交机密信息或有问题代码的风险有所增加。虽然已有许多代码验证机制，例如持续集成，但 pre-commit hooks 是一种简单且有效的防护措施，值得更多团队采用。不过，如果在 hooks 中加入运行缓慢的检查任务，可能会让开发者不愿意使用它们。因此，最好的做法是让这些 hooks 保持精简，专注于那些在此阶段最有效检测的风险，例如密钥扫描。

4. 使用生成式 AI 来理解遗留代码库

采纳

在最近几个月里，我们已经看到明确的证据表明 使用生成式 AI 来理解遗留代码库 可以显著加速对大型复杂系统的理解。诸如 Cursor、Claude Code、Copilot、Windsurf、Aider、Cody、Swimm、Unblocked 和 PocketFlow-Tutorial-Codebase-Knowledge 等工具帮助开发者发现业务规则、总结逻辑并识别依赖关系。它们与开放框架和直接结合 LLM 提示词一起使用，大大减少了理解遗留代码库所需的时间。

我们在多个客户中的经验表明，借助 GenAI 辅助理解遗留系统如今已成为一种实用的默认方案，而非实验性质。设置工作量因方法而异，特别是对于 GraphRAG 等高级方法，并且往往与正在分析的代码库的大小和复杂性成比例。尽管如此，其对生产力的影响是持续且显著的。生成式 AI 已经成为我们探索和理解遗留系统方式的重要组成。

5. AGENTS.md

试验

AGENTS.md 是一个为 AI 编码智能体在项目中提供操作指引的通用格式。本质上，它相当于面向智能体的 README 文件，除 Markdown 格式外没有强制的字段或特定格式，完全依赖于基于 LLM 的编码代理能够理解人类编写的、易于理解的指导。典型用法包括：如何使用编码环境中的工具、测试说明以及管理提交的最佳实践建议。虽然 AI 工具支持多种上下文工程方法，但 AGENTS.md 的价值在于创建了一个简单的文件规范作为起点。

6. AI用于代码迁移

试验

代码迁移的形式多种多样，从语言重写到依赖项或框架升级，几乎都不是一件小事，往往需要数月的人力投入。我们有一个团队在升级 .NET 框架版本时，尝试用 AI 缩短这一流程。过去，我们曾推荐 OpenRewrite 这类确定性、基于规则的重构工具。单靠 AI 自动升级通常代价高昂，且易陷入无效对话。于是，团队把传统升级流水线与智能编码助手结合，推进复杂的转型。区别于全程交由 AI 升级，他们将流程细化为可验证的小步骤：分析编译错误、生成迁移差异、反复验证测试。这种混合方法让 AI 编码智能体成为软件维护中的实用协作者。业界案例，如 Google 的大规模 int32 到 int64 迁移，也展现出同样趋势。尽管这一实践在节省时间方面结果好坏参半，但减少开发者重复劳动的潜力十分明显，值得持续探索。

7. Delta Lake liquid clustering

试验

Liquid clustering 是 Delta Lake 表的一种可作为分区和 Z 排序的替代方案的技术。在过去，优化 Delta 表的读取性能需要在表创建时基于预期的查询模式定义分区和 Z 排序键。后续修改这些键需要完整的数据重写。相比之下，聚类采用基于树的算法根据指定的键对数据进行聚类，可以增量更改而无需重写所有数据。这提供了更大的灵活性来支持多样化的查询模式，从而降低计算成本并增强读取性能。此外，Delta Lake 的 Databricks Runtime 通过分析历史查询工作负载、识别最优列并相应地聚类数据来支持自动 liquid clustering。独立的 Delta Lake 和 Databricks Runtime 用户都可以利用 liquid clustering 来优化读取性能。

8. 使用GenAI的自助式 UI 原型设计

试验

我们使用术语 使用 GenAI 的自助式 UI 原型设计 来描述一种新兴技术，其中工具如 Claude Code、Figma Make、Miro AI 和 v0 使产品经理能够直接通过文本提示生成交互式、可供用户测试的原型。团队无需手动绘制线框图，即可在几分钟内生成可运行的 HTML、CSS 和 JS 工件——具备草图般的速度，却拥有真实的交互性

和更高的保真度。这些“一次性”原型以快速学习为目的，牺牲了精细度，非常适合在设计冲刺的早期验证阶段使用。然而，较高的保真度可能导致团队过度关注细节或对上线所需工作量产生不切实际的期望——因此，明确的目标设定与期望管理至关重要。

当与用户研究结合使用时，该技术能通过将抽象想法转化为可感知的体验来加速探索过程。然而，团队应谨慎，避免让这些工具取代研究过程本身。如果运用得当，自助式原型设计能缩短反馈循环、降低非设计人员的参与门槛，并帮助团队在速度与质量之间实现健康平衡。

9. 从 LLMs 获取结构化输出

试验

从 LLMs 获取结构化输出 是指通过对大语言模型进行约束，使其输出符合预定义格式（如 JSON 或特定的编程类）。这一技术对构建可靠、生产级应用至关重要，它能将 LLM 通常不可预测的文本转化为机器可读、确定性的数据契约。基于我们的生产实践成果，我们将该技术从“评估”阶段提升为“试用”阶段。

相关方法覆盖从简单的提示词格式化、模型原生结构化输出，到更为健壮、借助 Outlines 和 Instructor 等工具实现的约束解码方式，这些工具通常通过有限状态机来确保输出的合法性。我们已经成功运用这一技术，从各类文档中提取复杂的非结构化数据并转换为结构化的 JSON，以便后续业务逻辑处理。

10. 测试 && 提交 || 回退 (Test && Commit || Revert)

试验

测试 Test && 提交 Commit || 回退 Revert（TCR）是一种源自测试驱动开发的编程工作流，通过一个简单规则促进非常小的、连续的步骤：每次修改后，如果测试通过，则提交更改；如果测试未通过，则回退更改。实现 TCR 非常直接：只需定义一个脚本，在代码库中自动执行此循环。TCR 最初由 Kent Beck 在一篇经典文章中提出。我们发现 TCR 强化了诸如 YAGNI 和 KISS 等积极的编码实践。在我们尝试使用生成式 AI 构建软件的新工作流时，这值得评估。

11. AI 驱动的 UI 测试

评估

在上一期技术雷达中，AI 驱动的 UI 测试 主要集中于探索性测试，我们当时指出 LLM 的非确定性可能会引入不稳定性。随着 MCP 的兴起，我们看到主流的 UI 测试框架如 Playwright 和 Selenium 正在引入各自的 MCP 服务器（playwright-mcp、mcp-selenium）。这些服务器通过其原生技术提供可靠的浏览器自动化，使编码助手能够在 Playwright 或 Selenium 中生成更可靠的 UI 测试。尽管 AI 驱动的 UI 测试仍是一个快速演进的领域——例如最新版本的 Playwright 已引入 Playwright Agents——但我们对这些进展感到振奋，并期待看到更多实践经验与落地指南的出现。

12. 将编码智能体锚定到参考应用

评估

过去我们曾在雷达上提到过定制服务模板模式，该模式通过提供合理的默认值来引导新服务并与现有基础设施无缝集成，帮助组织部署微服务。然而，随着时间的推移，随着新依赖项、框架和架构模式的出现，这些模板与现有服务之间的代码漂移往往会增长。为了保持良好的实践和架构一致性——特别是在编码智能体团队时代——我们一直在试验将编码智能体锚定到参考应用。这种模式通过提供实时的、可编译的参考应用而不是静态提示示例来指导生成式代码智能体。模型上下文协议（MCP）服务器暴露参考模板代码和提交差异，使智能体能够检测漂移并提出修复建议。这种方法将静态模板转换为 AI 可以智能引用的活的、适应性强的蓝图——在系统演进过程中保持一致性、减少分歧并改善对 AI 驱动脚手架的控制。

13. 上下文工程

评估

上下文工程 是在推理过程中，对提供给大语言模型的信息进行系统性设计与优化，以稳定可靠地产出期望结果。它涉及对上下文要素的结构化、选择与编排——例如提示词、检索数据、记忆、指令以及环境信号——以便让模型的内部层处于最优状态。不同于只关注提示措辞的提示工程，上下文工程关注的是上下文的整体配置：即如何组织与传递相关知识、指令以及先前上下文，以实现最有效的结果。

当下，工程师采用的多种不同技术大体可分为三个方面：上下文设置 涵盖诸如使用最小系统提示词、规范的少样本示例以及令牌高效的工具等策略，用于决定性行动。针对长周期任务的上下文管理 通过上下文摘要 和 结构化笔记 来持久化长期记忆，并通过子代理架构 来隔离和总结复杂的子任务，从而应对有限的上下文窗口。动态信息检索 依赖于即时（JIT）上下文检索，其中智能体仅在需要时才自主加载外部数据，从而最大化效率与准确性。

14. 用于前向工程的生成式 AI

评估

用于前向工程的生成式 AI 是一种新兴技术，通过 AI 生成的遗留代码库描述来现代化遗留系统。它引入了一个明确步骤，专注于遗留代码做了什么（即其规格说明），同时刻意隐藏当前的实现方式。这与规范驱动开发相关，但特别应用于遗留系统的现代化。

通过在重写代码之前生成和迭代功能描述，团队可以利用生成式 AI 揭示隐藏的逻辑、依赖关系和可能被忽略的边界情况。强调问题空间而非现有系统，也允许生成式 AI 模型探索更具创造性和前瞻性的解决方案。该工作流遵循反向工程 → 设计 / 解决方案 → 前向工程的循环，使人类和 AI 智能体在提交实现前能够在更高层次进行推理。在 Thoughtworks，我们看到多个团队成功应用这一方法来加速遗留系统的重写。其目标并非完全隐藏实现细节，而是引入一个临时抽象，帮助团队和智能体在不受现有结构限制的情况下探索替代方案。这一技术在生成更清晰、更易维护和面向未来的代码方面显示出良好前景，同时也减少了理解现有实现所需的时间。

15. 将 GraphQL 作为 LLM 的数据访问模式

评估

将 GraphQL 作为 LLM 的数据访问模式 是一种新兴方法，用于创建统一、适合模型的数据访问层，从而增强上下文工程。它使团队能够暴露结构化、可查询的数据，而无需授予模型直接访问数据库的权限。与通常会过度获取数据或需要为每个用例创建新端点或过滤器的 REST API 不同，GraphQL 允许模型仅获取所需数据——减少噪音、提高上下文相关性并降低 token 使用量。

定义良好的 GraphQL 架构还提供元数据，使 LLM 能够推理可用实体和关系，从而支持智能体用例的动态、架构感知查询。这种模式在 REST 与 SQL 之间提供了安全的中间方案，在治理控制与灵活访问之间取得平衡。

然而，该方法依赖于结构良好的架构和有意义的字段名称。解释架构语义和处理复杂结构仍然具有挑战性——对人类难以推理的内容通常对 LLM 也同样困难。同时，需要注意增加的 DoS 攻击向量，以及 GraphQL 常见挑战，如缓存和版本管理。

16. 知识流量胜于知识存量

评估

我们经常被问到一个问题：“如何改进团队之间的信息共享方式？”管理组织知识的技术在不断演进，我们发现一个借鉴自系统思维的有价值视角: 知识流量与知识存量。这一框架源自经济学，鼓励团队将组织知识视为一个系统——存量代表累积的知识，流量代表知识在组织中如何流动与演化。增加外部知识流入往往会提升创新。一种经受时间考验的提升流动性的方式是建立实践社区，它一贯显示出可量化的收益。另一种方法是有意识地寻求多样化的外部洞见来源。随着生成式 AI 工具让现有的知识存量更易获取，我们需要记住，培育新思想和外部视角与采用新技术同样关键。

17. 将 LLM 用作评审

评估

使用 LLM 作为评审——评估另一个系统（通常是基于 LLM 的生成器）的输出——因其在生成式 AI 中提供可扩展、自动化评估的潜力而备受关注。然而，为了反映新近发现的复杂性和风险，我们将此讨论从“试验”阶段移至“评估”阶段。尽管该技术提供速度和规模，但它常常无法作为人类判断的可靠代理。评估容易受到位置偏差、冗长偏差和低稳健性影响。更严重的问题是规模污染: 当 LLM 作为评判者用于奖励建模的训练流程时，它会引入自我增强偏差（即模型族倾向于自身的输出）和偏好泄漏，从而模糊训练和测试之间的界限。

这些缺陷导致过拟合结果，使性能指标虚高但缺乏现实有效性。已有研究对这一模式进行了更严格的调查。为应对这些缺陷，我们正在探索改进技术，例如使用 LLM 作为陪审团（通过多个模型达成共识）或在评估过程中使用链式思维推理。尽管这些方法旨在提高可靠性，但也增加了成本和复杂性。我们建议团队谨慎对待此技术——在将 LLM 评审用于关键工作流前，确保有人类验证、透明性和伦理监管。该方法仍然有效，但不如之前认为的那样成熟。

18. 设备端信息检索

评估

设备端信息检索 是一种使搜索、上下文感知和检索增强生成（RAG）能够完全在用户设备上运行——譬如移动设备、桌面设备或边缘设备，并优先考虑隐私和计算效率的技术。它将轻量级本地数据库与针对设备端推理优化的模型相结合。一个有前景的实现是将 sqlite-vec（一个支持嵌入式数据库内向量搜索的 SQLite 扩展）与 EmbeddingGemma（一个基于 Gemma 3 架构构建的 3 亿参数嵌入模型）进行配对。这种组合针对效率和资源受限环境进行了优化，将数据保持在边缘附近，从而减少对云 API 的依赖，并改善延迟和隐私。我们建议团队评估这种技术用于本地优先应用和其他数据主权、低延迟和隐私至关重要的场景。

19. SAIF

评估

SAIF（Secure AI Framework，安全 AI 框架）是由 Google 开发的一套框架，旨在为管理 AI 安全风险提供实用指南。它通过清晰的风险地图、组件分析和可操作的缓解策略，系统性地应对常见威胁，如数据投毒和提示注入。我们认为，SAIF 对构建智能体系统过程中不断演变的风险的关注，恰逢其时且极具价值。SAIF 提供了一份简明且可执行的操作手册，帮助团队强化 LLM 使用及 AI 驱动应用的安全实践。

20. 无边车服务网格

评估

随着基于 sidecar 的服务网格在成本和运维复杂性方面的持续存在，我们很高兴看到另一种 无边车服务网格（Service mesh without sidecar）方案的出现：Istio ambient 模式。Ambient 模式引入了一种分层架构，将职责划分为两个关键组件：每个节点的 L4 代理（ztunnel）和每个命名空间的 L7 代理（Waypoint proxy）。ztunnel 确保 L3 和 L4 流量能够高效且安全地传输。它通过为所有节点身份获取证书并处理往返于启用 Ambient 模式工作负载的流量重定向，从而支撑整个 Ambient 数据平面。Waypoint proxy 是一个可选的 Ambient 模式组件，可启用更丰富的 Istio 功能，如流量管理、安全性和可观测性。我们在小规模集群中已经获得了良好的使用体验，并期待随着采用的增长，能在大规模集群中获得更多实践洞察和最佳实践。

21. 小语言模型(SLMs)

评估

我们观察到 小语言模型 (SLMs) 在多个版本的科技雷达中稳步发展。随着对构建智能体解决方案的兴趣不断增长，我们看到了越来越多的证据表明 SLMs 可以高效地支持智能体 AI。目前大多数智能体工作流都集中在狭窄、重复的任务上，不需要高级推理，这使得它们与 SLMs 非常匹配。SLMs 的持续进展，如 Phi-3、SmolLM2 和 DeepSeek，表明 SLMs 在这些任务中提供了足够的功能——与 LLMs 相比，具有更低成本、更低延迟和更低资源消耗的好处。值得考虑 SLMs 作为智能体工作流的默认选择，只在必要时保留更大、更资源密集的 LLMs。

22. 规范驱动开发（Spec-driven development）

评估

规范驱动开发（Spec-driven development）是 AI 辅助编码流程中的一种新兴方法。虽然该术语的定义仍在不断演化，但一般而言，它指的是以结构化的功能规范为起点的开发流程，随后通过多个步骤将规范拆解为更小的部分、方案和任务。规范本身可以有多种形式：可以是一份文档、一组文档，或是以结构化工件捕捉不同功能方面的信息。

我们看到许多开发者已经采用了这种风格（我们在 Thoughtworks 也有一套内部分享的规范）。最近有三款工具分别探索了规范驱动开发的不同实现方式：亚马逊的 Kiro 引导用户经历三个工作流阶段——需求、设计和任务创建；GitHub 的 spec-kit 也采用了类似的三步过程，但增加了更丰富的编排、可配置的提示词以及一份“宪章”，定义了始终必须遵循的不可变原则；Tessl Framework（截至 2025 年 9 月仍处于内测阶段）则采纳了一种更为激进的方式，将规范本身而非代码作为维护对象。

我们认为这一领域极具吸引力，尽管目前的开发流程依然较为繁琐且带有很强的主观性。这些工具的行为因任务的规模和类型而迥异：有些工具会生成冗长难以审阅的规范文件；当它们产出 PRD 或用户故事时，有时也不清楚目标受众是谁。我们也许正在重蹈惨痛教训——为人工智能手工编写详细的规则最终无法实现规模化。

23. 编码智能体团队

评估

编码智能体团队 指的是开发者协同编排多个具备不同角色的 AI 编码智能体（例如架构师、后端专家、测试人员等）共同完成开发任务的一种技术实践。这一做法已被 Claude Code、Roo Code、Kilo Code 等工具支持，它们提供了子代理（subagents）机制和多种运行模式。基于给大型语言模型（LLM）分配具体角色和身份能够提升输出质量的成熟理念，该方法旨在通过协调多个具备特定职能的智能体（而非依赖单一通用型智能体）以获得更佳的开发成果。当前，最优的智能体角色划分方式仍在探索中；这种方法甚至可以突破传统团队角色的一对一映射。这一思路标志着业界正向更加可编排、细致分工的多步骤 AI 辅助开发流程转变。

24. 拓扑感知调度

评估

GPU 和 LPU 不再是独立设备，而是紧密耦合的加速器网络，其性能取决于放置位置和拓扑。在 NVIDIA 的 NVL72 等机架级系统中，72 个 GPU 共享超过 13 TB 的显存，并作为单一加速器运行——直到工作负载跨交换机网络，集体操作才会成为瓶颈。类似地，Groq 的编译时、软件调度架构假设数据移动是确定性的；随机调度会破坏这些假设和可预测性。即便在同一数据中心内，GPU 性能也可能存在显著差异，这就产生了对拓扑感知调度的需求，在作业放置时同时考虑硬件布局和性能波动。

忽略 NVLink、PCIe 或 NIC 拓扑的简单调度器，往往会随意分散多 GPU 工作负载，导致步骤时间和效率下降。训练工作负载是同步且带宽受限的，更适合在连续的 NVLink 网络上调度，确保所有 reduce 和流水线阶段拥有统一、高带宽路径。这些作业应基于互连带宽进行协同调度，避免跨交换机跳转，并将链路、交换机和节点边界视为故障域。相比之下，推理工作负载受延迟和 SLO 限制，通常在跨域高可用复制与分片之间平衡，以保持专家混合（MoE）和 KV 缓存的局部性在最短路径上。针对预填充与解码阶段、微批处理以及租户隔离优化

放置，可进一步提升效率。我们认为，随着加速器性能越来越依赖网络和数据中心拓扑，拓扑感知调度将成为必需。我们的团队已在评估 Kueue 及相关项目，以提高放置精度、提升性能并确保客户的可靠扩展。

25. AI 有害流程分析

评估

现在广为流传的玩笑——MCP 中的 S 代表“安全”——代表了一个非常真实的问题。当智能体通过工具调用或 API 调用彼此通信时，它们可能很快遭遇被称为致命三威胁的情况：访问私有数据、接触不可信内容，以及能够进行外部通信。具备这三项的智能体极易受到攻击。由于 LLM 倾向于遵循输入中的指令，包含向不可信源导出数据指令的内容很容易导致数据泄露。一种新兴的缓解风险技术是有害流程分析，它通过检查智能体系统的流程图来识别潜在不安全的数据路径，以便进一步调查。虽然仍处于早期阶段，但有害流程分析代表了若干有前景的方法之一，用于检测智能体系统和 MCP 服务器日益暴露的新攻击向量。

26. AI 加速影子 IT（AI-accelerated Shadow IT）

暂缓

AI 正在降低非专业开发人员自行构建和集成软件的门槛，使他们无需等待 IT 部门响应自己的需求。尽管我们对这种技术带来的潜力感到兴奋，但同时也开始关注到 AI 加速影子 IT（AI-accelerated Shadow IT）的初步迹象。一些无代码（No-code）工作流自动化平台已支持对 AI API（如 OpenAI 或 Anthropic）的集成，这使得用户可能倾向于将 AI 用作“胶带”，将此前难以实现的系统集成临时拼凑起来，例如通过 AI 将聊天消息转换为 ERP 系统的 API 调用。同时，越来越多具有自主 Agent 能力的 AI 编码助手，甚至允许仅经过基础培训的非技术人员创建内部工具应用。

这些迹象呈现出类似于电子表格（Spreadsheets）当年迅速扩散的特征：虽然为企业关键流程提供了快速解决方案，但在长期运行后往往会造成规模更大的技术债（Tech Debt）。如果不加管控，这种新型影子 IT 将导致未经治理的应用程序激增，安全隐患加剧，数据分散在不同系统内。我们建议企业对此风险保持警觉，并谨慎评估快速问题解决与长期技术稳定性之间的平衡与取舍。

27. 容量驱动开发

暂缓

现代软件开发实践成功的关键在于保持对工作流的关注。流对齐团队专注于单一、有价值的工作流——例如一个用户旅程或产品——从而能够高效地交付端到端价值。然而，我们注意到一个令人担忧的趋势，即出现了 容量驱动开发（Capacity-driven development）。在这种模式下，这些团队在有空余产能时，会接手来自其他产品或工作流的功能开发。虽然这在短期内看似高效，但实际上是一种局部优化，仅适用于应对突发的需求高峰。当这种做法被常态化后，会增加认知负荷和技术债务；在最糟糕的情况下，随着跨产品上下文切换的成本不断累积，可能导致整体的拥塞崩溃。拥有空余产能的团队更应专注于提升系统健康度。为了更有效地管理产能，可以通过设置 WIP 限制来控制相邻工作流之间的工作量；在高需求时期考虑跨技能培训；并在必要时采用动态重组团队等技术手段。

28. 自满于 AI 生成的代码

暂缓

我们看到诸多数据与研究表明，随着 AI 编码助手和智能体的普及，人们对 自满于 AI 生成的代码 的担忧也在加剧。尽管有充分证据显示，这类工具可显著加快开发速度——尤其是在原型开发和全新项目场景中——但研究结果也表明，随着时间推移，代码质量可能会出现下降。

GitClear 2024 年的研究发现，重复代码和代码的反复变动比预期增长更多，而提交历史中的重构活动则有所下降。类似趋势也出现在微软研究中：知识型员工因 AI 助手而增强的自信，往往以批判性思维能力下降为代价——我们也注意到，长期使用编码助手后容易出现惰性和过度依赖的现象。

随着编码智能体推动更大范围的自动化变更，这一风险也被放大，因为由 AI 生成的更大代码变更集更加难以评审。正如任何系统一样，流水线中的某一环提速后，其他环节的压力也会随之增加。我们的团队发现，要在生产环境中高效安全地使用 AI，必须重新关注和强化代码质量。我们建议继续落实如 TDD（测试驱动开发）和静态分析等成熟实践，并将这些措施直接集成进开发流程，例如通过为软件团队精选共享指令等方式实现。

29. 天真的API到MCP转换

暂缓

许多组织都希望让 AI 智能体能够与现有系统交互，常见的做法是尝试将内部 API 直接无缝转换为 模型上下文协议（MCP）。越来越多的工具，如 MCP link 和 FastAPI-MCP，都在支持这种转换。

然而我们并不建议直接进行这种天真的 API 到 MCP 转换。API 通常是为人类开发者设计的，往往包含许多细粒度、原子性的操作 如果让 AI 代理串联调用这些操作，可能会导致 Token 过度消耗、上下文污染，以及智能体性能下降。此外，这些 API（尤其是内部接口）常常会暴露敏感数据或允许执行危险操作。对于人类开发者，此类风险可以通过架构模式和代码审核来缓解，但如果将 API 直接暴露给通过 MCP 协议集成的智能体，则无法可靠地阻止自治 AI 智能体误用这些接口。因此，我们建议针对智能体工作流，专门设计并构建安全的 MCP 服务器架构，在现有 API 基础上进行适配。

30. 独立数据工程团队

暂缓

组织独立数据工程团队来开发和拥有数据管道和产品——与它们服务的流对齐业务域分离——是一种会导致效率低下和业务成果薄弱的反模式。这种结构重复了过去隔离 DevOps、测试 或 部署 功能的错误，创造了知识孤岛、瓶颈和浪费的精力。若没有密切协作，数据工程师往往缺乏设计有意义的数据产品所需的业务和领域上下文，限制了采用的可能性和价值。相比之下，数据平台团队应该专注于维护共享基础设施，而由跨职能业务团队遵循数据网格原则构建和落地他们的数据产品。我们将这种做法置于“暂缓”状态，旨在劝退各自为政的组织模式——尤其是在对领域信息丰富、可用于人工智能的数据的需求持续增长的情况下。

31. Text to SQL

暂缓

Text to SQL 利用大语言模型（LLM）将自然语言翻译为可执行的 SQL，但其可靠性往往低于预期。我们已将此技术的状态调整为“Hold”，以劝阻在无人监督的工作流中使用——例如，动态转换用户生成的查询，而输出结果被隐藏或自动化。在这些情况下，LLM 由于对数据库模式或领域理解有限，常常会产生幻觉，导致错误的数据检索甚至意外的数据修改。此外，LLM 输出的非确定性特征也使得调试和审计错误变得更加困难。

我们建议 Text to SQL，并要求对所有生成的查询进行人工审核。对于智能化商业分析场景，应避免直接访问数据库，而应通过受治理的数据抽象语义层来实现，例如 Cube 或 dbt 的语义层；或者使用具有更强语义表达能力的访问层，如 GraphQL 或 MCP。

平台

采纳

32. 云上 Arm

试验

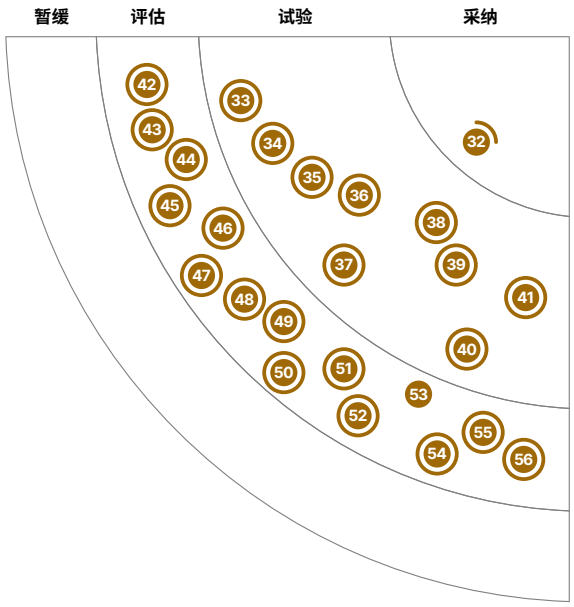
33. Apache Paimon
34. DataDog LLM Observability
35. Delta Sharing
36. Dovetail
37. Langdock
38. LangSmith
39. 模型上下文协议（MCP）
40. n8n
41. OpenThread

评估

42. AG-UI 协议
43. Agent-to-Agent (A2A) 协议
44. Amazon S3 Vectors
45. Ardoq
46. CloudNativePG
47. Coder
48. Graft
49. groundcover
50. Karmada
51. OpenFeature
52. Oxide
53. Restate
54. SkyPilot
55. StarRocks
56. Uncloud

暂缓

—



新的　挪进 / 挪出　没有变化

32. 云上 Arm

采纳

云端 Arm 计算实例近年来因其相较传统 x86 实例更具成本效益和能效，受到了越来越多的关注。主流云服务商——包括 AWS、Azure 和 GCP——如今都提供了强大的 Arm 方案。对于大规模或对成本敏感的工作负载，这类实例尤其具备吸引力。我们的多个团队已顺利将微服务、开源数据库，甚至高性能计算等工作负载迁移至 Arm 平台，只需极少的代码修改和轻微的构建脚本调整。越来越多新兴云原生应用和系统，也默认选择 云端 Arm 作为首选架构。根据我们的实践经验，除非有特殊的架构依赖，我们推荐大多数工作负载优先考虑 Arm 计算实例。现代工具（如 多架构 Docker 镜像）也进一步简化了 Arm 与 x86 环境的构建与部署流程。

33. Apache Paimon

试验

Apache Paimon 是一种开源数据湖格式，旨在支持数据湖仓一体架构。它可与 Flink 和 Spark 等处理引擎无缝集成，支持流处理与批处理操作。Paimon 架构的一大核心优势在于将标准数据湖格式与 LSM（日志结构合并树）结构相融合。这一组合解决了数据湖在高性能更新和低延迟读取方面的传统难题。Paimon 支持主键表以实现高吞吐、实时更新，并内置可定制的合并引擎，支持去重、部分更新和聚合。这一设计使得在数据湖中直接进行高效的流数据写入及可变状态管理成为可能。除此之外，Paimon 还提供成熟的数据湖能力，包括可扩展的元数据、ACID 事务、时间穿梭、模式演进，以及通过压缩与 Z-ordering 实现的优化数据布局。我们建议在需要统一存储层以实现高效处理大规模追加式数据和复杂实时流更新的项目中评估使用 Paimon。

34. DataDog LLM Observability

试验

DataDog LLM Observability 为大语言模型和智能体应用工作流提供端到端的跟踪、监控和诊断。它将每个提示、工具调用和中间步骤映射到跨度和跟踪中；跟踪延迟、令牌使用、错误和质量指标；并与 Datadog 更广泛的 APM 和可观测性套件集成。

对于已经使用 Datadog 并熟悉其成本结构的组织而言，如果这些工作负载可以进行插桩，LLM 可观测性功能可能是一种直接了解 AI 工作负载的方法。然而，配置和使用 LLM 检测需要谨慎和对工作负载及其实现的深入理解。我们建议数据工程师和运维人员在部署时密切合作。另请参阅我们关于避免独立数据工程团队的建议。

35. Delta Sharing

试验

Delta Sharing 是一个用于安全与跨平台数据共享的开放标准和协议，由 Databricks 和 Linux Foundation 开发。它与云平台无关，使组织能够在云提供商和本地位置之间共享实时数据，而无需复制或重复数据，从而保持数据的新鲜度并消除重复成本。我们看到一家电子商务公司成功使用 Delta Sharing 将分散的合作伙伴数据共享系统替换为集中式、实时和安全的平台，显著改善了协作。该协议使用简单的 REST API 来颁发短期预签名 URL，允许接收者使用 pandas、Spark 或 Power BI 等工具检索大型数据集。它支持共享数据表、视图、AI 模型和笔记本。虽然它提供了强大的集中治理和审计功能，但用户仍应注意云出站费用，如果管理不当，这可能成为重大的运营风险。

36. Dovetail

试验

Dovetail 解决了长期困扰团队的零散定性调研数据管理难题。它为用户访谈、转录和洞察提供集中存储，将原始数据转化为结构化、可分析的资产。在产品探索流程中，我们发现 Dovetail 能帮助建立清晰的证据链，从用户语录和提炼主题直接关联到产品假设和 ROI 预估，从而增强了定性数据在产品决策中的作用。

37. Langdock

试验

Langdock 是一个供组织开发和运行生成式 AI 智能体和工作流以进行内部运营的平台。它提供了一个统一的环境，包括内部聊天助手、用于连接多个 LLM 的 API 层，以及用于构建与 Slack、Confluence 和 Google Drive 等系统集成的智能体工作流的工具。该平台强调数据主权，提供本地部署和欧盟托管选项，并符合企业合规标准。

部署 Langdock 的组织仍应密切关注数据治理，并使用 AI 有害流程分析等技术来避免致命三威胁。使用者还应考虑平台的成熟度，评估他们需要的具体集成，并规划可能需要的任何自定义开发。

38. LangSmith

试验

LangSmith 是由 LangChain 团队推出的托管平台，用于为大型语言模型（LLM）应用提供可观测性、追踪和评估功能。它能够捕获链、工具和提示的详细追踪信息，帮助团队调试和分析模型行为、监控性能回退以及管理评估数据集。LangSmith 是一个专有的 SaaS 服务，对非 LangChain 工作流的支持有限，因此主要适用于已经深度使用 LangChain 生态的团队。与开源替代方案（如 Langfuse）相比，它在提示评估和实验集成方面的功能更加完善和成熟。

39. 模型上下文协议（MCP）

试验

模型上下文协议（Model Context Protocol，MCP）是一个开放标准，用于定义 LLM 应用程序和智能体如何与外部数据源和工具集成，从而显著提升 AI 生成输出的质量。MCP 专注于上下文和工具访问，这使其区别于负责智能体间通信的 Agent2Agent (A2A) 协议。它定义了服务器（用于访问数据库、Wiki 和服务等数据与工具）和客户端（如智能体、应用程序与代码助手）。自上次收录以来，MCP 的采用率迅速上升，越来越多的公司加入生态系统，包括 JetBrains（IntelliJ）和 Apple，以及新兴框架如 FastMCP。MCP 注册表预览版现已支持公共和专有工具的发现。然而，MCP 的快速演进也暴露出一些架构层面的不足，并因忽视了数十年的 RPC 最佳实践而遭到批评。在生产应用中，团队应超越炒作，加强审查与防护，例如使用 MCP-Scan 缓解有害数据流风险，并密切关注授权模块草案的安全进展。

40. n8n

试验

n8n 是一个公平代码许可的工作流自动化平台，类似于 Zapier 或 Make（前身为 Integromat），但专为希望获得自托管、可扩展和代码可控选项的开发者而构建。它提供了比 Apache Airflow 更低代码的可视化工作流创建方法，同时仍支持 JavaScript 或 Python 的自定义代码。

其主要用例是将多个服务集成到自动化工作流中，但它也可以将大语言模型与可配置的数据源、内存和工具连接起来。我们的许多团队使用 n8n 来快速构建由聊天应用程序或 Webhook 触发的智能体工作流原型，并经常利用其导入和导出功能来生成具有 AI 辅助的工作流。一如既往，我们建议在生产环境中使用低代码平台时要谨慎。不过 n8n 的自托管和代码定义的工作流可以缓解其中的一些风险。

41. OpenThread

试验

OpenThread 是 Google 开发的 Thread 网络协议的开源实现。它支持 Thread 规范的所有关键功能——包括 IPv6、6LoWPAN 和 LR-WPAN 等网络层，以及允许设备同时作为节点和边界路由器运行的网状网络功能。OpenThread 可在广泛的硬件平台上运行，它利用灵活的抽象层和集成钩子，使供应商能够整合他们自己的无线电和加密功能。这一成熟的协议已广泛应用于商业产品中，并且根据我们的经验，它已被证明是构建各种物联网解决方案的可靠之选，从电池供电的低功耗设备到大规模网状传感器网络，均可胜任。

42. AG-UI协议

评估

AG-UI 是一个开放协议和库，旨在规范丰富用户界面与智能体之间的通信。该协议专注于面向用户的智能体应用，并通过中间件和客户端集成实现前端与后端的通用交互。协议定义了后端智能体与前端应用交流的一致方式，使 AI 和人类用户之间能够进行实时、有状态的协作。它支持多种传输协议，包括 SSE 和 WebSockets，并提供标准化的事件类型，用于描述智能体执行的不同阶段。目前已内置兼容主流智能体框架，如 LangGraph 和 Pydantic AI，并有社区集成其他解决方案。

43. Agent-to-Agent (A2A) 协议

评估

Agent2Agent (A2A) 是一个协议，定义了复杂、多智能体工作流中智能体之间通信和交互的标准。它使用 Agent Cards 描述智能体间通信的关键元素，包括技能发现和传输及安全方案的规范。A2A 通过专注于智能体间通信，而不暴露内部细节（如智能体状态、记忆或内部），这成为了对模型上下文协议 (MCP) 的一个补充。

该协议提倡最佳实践，例如对长时间运行的任务采用异步优先方法，对增量更新采用流式响应，以及使用 HTTPS、身份验证和授权进行安全传输。为了方便快速采用，A2A 提供了 Python、JavaScript、Java 和 C# 的 SDK。尽管 A2A 相对较新，但它使团队能够构建特定领域的代理，这些代理可以协作形成复杂的工作流，使其成为此类场景的理想选择。

44. Amazon S3 Vectors

评估

Amazon S3 Vectors 为 S3 对象存储原生扩展了向量能力，提供内置的向量存储与相似度检索功能。它可与包括 Amazon Bedrock、OpenSearch 在内的 AWS 生态系统无缝集成，并支持基于元数据的过滤与通过 IAM 的治理等附加特性。尽管目前仍处于预览阶段且受限制与约束影响，我们认为它的价值主张极具吸引力。这种高性价比、易于访问的向量存储方案有望支持涉及大量数据、但对低延迟要求不高的多样应用场景。

45. Ardoq

评估

Ardoq 是一个企业架构（EA）平台，使组织能够构建、管理和扩展其架构知识库，从而更有效地进行未来规划。与容易产生漂移和孤岛化的传统静态文档不同，Ardoq 的数据驱动方法从现有系统中提取信息，创建动态知识图谱，并随着环境变化保持更新。我们发现其中一个特别有用的功能是 Ardoq Scenarios，它允许使用类似 Git 的分支与合并方法，直观地建模和定义未来的假设情景。对于希望推动架构转型的组织，应评估像 Ardoq 这样的专用 EA 平台，以判断其在简化和加速该过程中的潜力。

46. CloudNativePG

评估

CloudNativePG 是一个 Kubernetes Operator，用于简化在 Kubernetes 中托管和管理高可用 PostgreSQL 集群的过程。在 Kubernetes 上运行像 PostgreSQL 这样的有状态服务可能非常复杂，需要深入了解 Kubernetes 和 PostgreSQL 的复制机制。CloudNativePG 将整个 PostgreSQL 集群视为单个可配置的声明式资源，从而抽象了大部分复杂性。它使用原生流复制提供无缝的主 / 备架构，并开箱即用地包含高可用功能，包括自愈能力、自动故障切换以提升最一致的副本以及自动重建失败的副本。如果你计划在 Kubernetes 上托管 PostgreSQL，CloudNativePG 是一个可靠的起点。

47. Coder

评估

Coder 是一个平台，用于快速配置标准化编码环境，遵循我们之前描述的云上开发环境实践。与类似工具，如 Gitpod（现已重新命名为 Ona）和 GitHub Codespaces 相比，Coder 通过 Terraform 提供了对工作站定制的更大控制。它将工作站在您自己的基础设施上托管，无论是云中还是数据中心，而不是在供应商的服务器上。这种方法提供了更大的灵活性，包括运行 AI 编码代理和访问内部组织系统的能力。然而，这种灵活性也带来了一些权衡：设置和维护工作站模板需要更多精力，并且在代理工作流中管理数据安全风险方面承担更大的责任。

48. Graft

评估

Graft 是一款事务型存储引擎，专为在边缘和分布式环境中实现强一致性、高效率的数据同步而设计。它通过惰性复制（Lazy Replication）实现按需同步数据，通过部分复制（Partial Replication）最大限度地减少带宽消耗，并通过可序列化快照隔离来保证数据完整性。我们曾在雷达中提及 Electric，它针对类似用例，但 Graft 的独特之处在于它将对象存储转变为一个事务型系统，支持在不强制规定数据格式的情况下，对数据子集进行一致性的页面级更新。这使其非常适合作为本地优先应用的底层支持、管理复杂的跨平台同步，以及作为无服务器或嵌入式系统中无状态副本的后端。

49. groundcover

评估

groundcover 是一个云原生可观测性平台，将日志、追踪、指标和 Kubernetes 事件统一在一个面板中。它利用 eBPF 技术，无需编写任何代码即可捕获细粒度的可观测性数据——也就是说，无需在应用程序代码中插入代理或 SDK。groundcover 的 eBPF 传感器在每个监控集群中运行在专用节点上，独立于它观察的应用程序。关键功能包括深度内核级可见性、自带云（BYOC）架构用于数据隐私和数据量无关的定价模型，从而保证成本可预测。

50. Karmada

评估

Karmada（“Kubernetes Armada”）是一个可在多个 Kubernetes 集群、云和数据中心间编排工作负载的平台。许多团队当前通过 Flux 等 GitOps 工具或 ArgoCD 结合自定义脚本跨集群部署，因此一个专为此设计的解决方案备受欢迎。Karmada 利用原生 Kubernetes API，无需对已为云原生环境构建的应用进行任何更改。它为多云管理、高可用性、故障恢复和流量调度等场景，提供了先进的调度能力。

Karmada 仍然相对较新，因此评估团队所依赖功能的成熟度尤为重要。不过，作为一个 CNCF 项目，它发展势头迅猛，我们已有多个团队成功部署使用。需要注意的是，诸如跨集群的网络、状态与存储管理等领域并不在 Karmada 的功能范围内。大多数团队仍需配合 Istio 或 Linkerd 这类 服务网格 进行流量管理，并提前规划有状态工作负载和分布式数据的管理方案。

51. OpenFeature

评估

随着企业规模的扩大，特性开关管理通常变得日益复杂；团队需要一个能够超越最简单的特性开关的抽象层。OpenFeature 通过一个与供应商无关、由社区驱动的 API 规范，提供了这样的抽象层。该规范标准化了特性开关的定义与消费方式，将应用代码与管理方案解耦。这样的灵活性让团队可以方便地切换供应商——无论是使用环境变量或内存配置等基础方案，还是像 ConfigCat 或 LaunchDarkly 这样成熟的平台。不过，仍然有一个关键的提醒：各团队必须对不同类型的开关进行有纪律的分别管理，以避免开关泛滥、应用复杂度增加和过多的测试负担。

52. Oxide

评估

构建和运营私有基础设施非常复杂，这也是公共云成为大多数组织默认选择的主要原因之一。然而，对于有这类需求的团队，Oxide 提供了一种替代方案，无需从零组装和集成硬件与软件。它提供预构建的机架，包括计算、网络和存储，并运行完全集成的系统软件。团队可以通过 Oxide 的 IaaS API 使用 Terraform 等自动化工具管理资源——Oxide 称之为 本地弹性基础设施。

戴尔和 VMware 的 VxRail、Nutanix 以及 HPE SimpliVity 也提供超融合基础设施（HCI）解决方案，但 Oxide 的独特之处在于其专用设计方法。它从电路板、电源到固件设计整个堆栈，而不是从不同供应商处组装组件。Oxide 还开发并开源了 Hubris，这是一个为嵌入式系统设计的轻量级、内存保护、消息传递内核，采用 Rust 编写，同时还有其他基于 Rust 的基础设施项目。我们还赞赏 Oxide 的设备和软件销售不收取许可费。

53. Restate

评估

Restate 是一个持久化执行平台，旨在解决构建有状态、容错应用程序时的复杂分布式系统挑战。它通过执行日志记录记录每个步骤，确保容错、可靠恢复和跨服务的精确一次通信。平台的关键架构优势在于将应用程序逻辑分离为三种持久化服务类型：基本服务用于无状态函数；虚拟对象用于建模并发、有状态的实体；工作流用于编排复杂、多步骤的过程。我们在一个大型保险系统中仔细评估了 Restate，对其性能感到非常满意。

54. SkyPilot

评估

SkyPilot 是一个开源平台，用于在本地或云端运行和扩展 AI 工作负载。由加州大学伯克利分校的 Sky Computing Lab 开发，SkyPilot 充当智能中介，能够自动寻找并分配主要云服务和 Kubernetes 集群中最便宜、最可用的 GPU，从而常常降低计算成本。对于基础设施团队而言，它通过提供类似 Slurm 的易用性、云原生的稳健性、对 Pod 的直接 SSH 访问，以及帮派调度和多集群支持等功能，简化了在 Kubernetes 上运行 AI 的流程，实现训练或推理工作负载的无缝扩展。

55. StarRocks

评估

StarRocks 是一个分析型数据库，通过结合传统 OLAP 系统的速度与现代 Lakehouse 架构的灵活性，重新定义了实时商业智能。它通过 SIMD 优化的执行引擎、列存储以及复杂的基于成本的优化器，实现了大规模环境下的亚秒级查询延迟。这种高性能架构允许用户直接在开放数据格式（如 Apache Iceberg）上运行复杂分析，无需预计算或复制数据。尽管该领域存在许多平台，我们认为 StarRocks 是需要极高并发性和实时数据一致性的成本效益解决方案的有力候选。

56. Uncloud

评估

Uncloud 是一款轻量级的容器编排与集群管理工具，旨在帮助开发者将 Docker Compose 应用快速部署到生产环境。它提供了一种简化的云端体验，同时避免了 Kubernetes 带来的复杂运维负担。为实现跨机器扩展和零停机部署，它自动配置安全的 WireGuard 网状网络以确保通信，并利用 Caddy 反向代理提供自动化 HTTPS 和负载均衡服务。Uncloud 的核心架构优势在于其彻底的去中心化设计，这种设计消除了对中心控制平面的依赖，确保即使集群中的单台机器离线，服务和集群操作仍能维持功能。借助 Uncloud，用户可以灵活地将云虚拟机和裸金属服务器整合为一个统一且节省成本的计算环境。

工具

采纳

57. ClickHouse
58. NeMo Guardrails
59. pnpm
60. Pydantic

试验

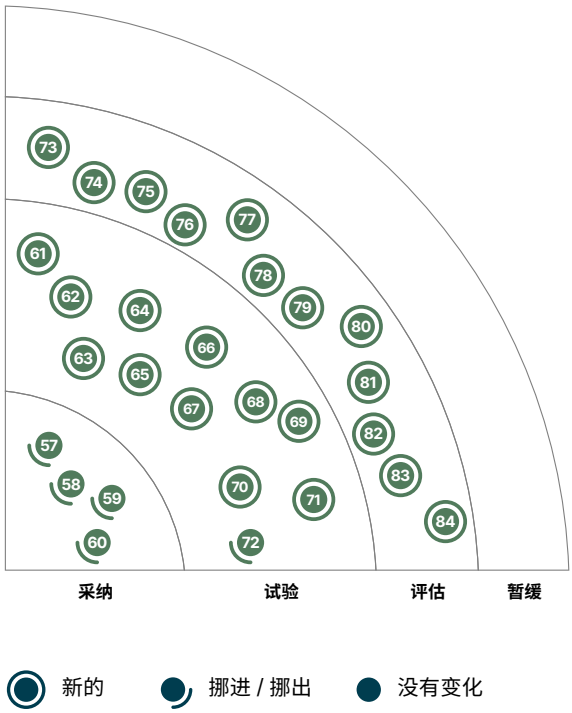
61. AI Design Reviewer
62. Barman
63. Claude Code
64. Cleanlab
65. Context7
66. 数据契约 CLI
67. Databricks Assistant
68. Hoppscotch
69. NVIDIA DCGM Exporter
70. RelationalAI
71. UX Pilot
72. v0

评估

73. Augment Code
74. Azure AI 文档智能
75. Docling
76. E2B
77. Helix 编辑器
78. Kueue
79. MCP-Scan
80. oRPC
81. Power user for dbt
82. Serena
83. SweetPad
84. Tape/Z（用于 Z/OS 的汇编程序分析工具集）

暂缓

—



57. ClickHouse

采纳

ClickHouse 是一个开源、分布式的列式在线分析处理（OLAP）数据库，适用于实时分析。它已经发展成为高性能、可扩展的数据分析引擎，能够处理大规模数据分析任务。其增量物化视图、高效查询引擎和强大的数据压缩功能，使其非常适合交互式查询。内置的近似聚合函数支持在精度与性能之间折衷，特别适用于高基数分析。S3 存储引擎和 MergeTree 的加入，实现了存储与计算的分离，可通过 S3 兼容存储托管 ClickHouse 表。我们还发现 ClickHouse 是 OpenTelemetry 数据以及如 Sentry 这类崩溃分析工具的优秀后端选择。对于希望获得快速、开源分析引擎的团队来说，ClickHouse 是极佳选择。

58. NeMo Guardrails

采纳

NeMo Guardrails 是 NVIDIA 的开源工具包，可以轻松为基于 LLM 的对话应用添加可编程的安全和控制机制。它通过定义和执行行为规则来确保输出保持安全、切题和合规。开发者使用 Colang（一种专用语言）来创建灵活的对话流程和管理对话，执行预定义的路径和操作程序。NeMo Guardrails 还提供了异步优先的 API 以获得更好的性能，并支持内容安全、安全性和输入输出审核的保障措施。我们看到，从简单的聊天机器人到复杂的代理工作流，各个团队都在稳步采用 NeMo Guardrails 构建各种类型的应用程序。随着其功能集的扩展和对常见 LLM 漏洞的成熟覆盖，我们决定将 NeMo Guardrails 移至采用评级。

59. pnpm

采纳

自上次技术雷达发布以来，我们持续收到团队对 pnpm 的积极反馈。pnpm 是一个 Node.js 的包管理器，相比其他替代方案，它在速度和磁盘空间效率方面都有显著提升。它通过将多个项目的 node_modules 文件夹中重复的包以硬链接（hard link）的方式指向磁盘上的单一位置，从而节省空间；同时还支持基于文件级的增量优化，进一步提升性能。由于 pnpm 能够在几乎没有兼容性问题的情况下提供更快的反馈循环，它已成为我们在 Node.js 包管理中的默认选择。

60. Pydantic

采纳

Pydantic 是一个 Python 库，使用标准类型提示来定义数据模型并在运行时强制执行数据模式。最初，Python 添加类型注解是为了静态分析，但其日益多样化的用途已扩展到运行时验证。基于高性能的 Rust 核心，Pydantic 提供高效的数据验证、解析和序列化。

虽然 Pydantic 在 Web API 开发中最为人知，但它在 LLM 应用中也变得不可或缺。我们通常使用从 LLMs 获取结构化输出技术来管理 LLM 的不确定性。通过定义严格的数据模式，它为模型输出的不确定性提供了安全网——将自由格式的文本响应转换为确定性、类型安全的 Python 对象（例如 JSON）。这种方法通常通过 Pydantic AI 或 LangChain 实现，将潜在脆弱的 LLM 交互转化为可靠的、机器可读的数据契约。我们的团队已成功在生产环境中使用 Pydantic 从非结构化文档中提取结构化表示，确保输出符合有效结构。鉴于其成熟度、性能和可靠性，Pydantic 现在是我们生产级 Python AI 应用的默认选择。

61. AI Design Reviewer

试验

AI Design Reviewer 是一个 Figma 插件，用于进行设计审计或启发式评估，并收集对现有或新设计的可操作反馈。其审计涵盖 UX 批评、UI 不一致性、可访问性差距、内容质量和边缘案例场景。除了识别问题外，它还提供领域感知的建议，帮助团队建立共享的设计词汇和设计选择背后的理由。我们的团队使用 AI Design Reviewer 来分析遗留设计——识别要保留的积极体验和要解决的负面体验——这为重新设计的 UX 目标提供了信息。它还可以作为同行评审的替代品，在开发交接之前为新设计提供早期反馈。

62. Barman

试验

Barman（Backup and Recovery Manager）是一个用于管理 PostgreSQL 服务器备份和灾难恢复的开源工具。它支持完整的灾难恢复流程，通过多种方法简化物理备份的创建，将备份组织到一个全面的目录中，并支持将备份恢复到运行中的服务器，同时具备时间点恢复功能。我们发现 Barman 功能强大且易于使用，并对迁移活动中时间点恢复操作的速度印象深刻。它也适用于定期备份，能够处理复杂的混合调度和保留配置。

63. Claude Code

试验

Anthropic 的 Claude Code 是一款具备智能体特性的 AI 编码工具，它为规划和执行复杂的多步骤工作流提供了自然语言接口和智能体执行模型。自发布不到一年以来，它已经在 Thoughtworks 内部和外部的开发者中被广泛采用，因此我们将其纳入 Trial 阶段。基于命令行的编码智能体，如 OpenAI 的 Codex CLI、Google 的 Gemini CLI 以及开源项目 OpenCode 相继推出，而基于 IDE 的助手，如 Cursor、Windsurf 和 GitHub Copilot 也纷纷加入了智能体模式。即便如此，Claude Code 仍然深受欢迎。我们观察到，团队不仅使用它来编写和修改代码，还将其作为通用型 AI 智能体，用于管理规格说明、用户故事、配置、基础设施以及文档。

智能体式编码将开发者的关注点从“编写代码”转移到“表达意图并委派实现”。这虽然能够加快开发周期，但也可能导致对 AI 生成代码的自满，从而产生人类和 AI 智能体都更难维护和演化的代码。因此，团队必须严格管理 Claude Code 的使用方式，可采用诸如上下文工程、精选共享指令以及可能的编码智能体团队等技术手段来实现。

64. Cleanlab

试验

在以数据为中心的 AI 范式中，改善数据集质量通常比调整模型本身带来更大的性能提升。Cleanlab 是一个开源 Python 库，旨在通过自动识别常见的数据问题来解决这一挑战——如存在于文本、图像、表格和音频数据集之中错误标签、异常值和重复项。基于置信学习原理构建，Cleanlab 利用模型预测的概率来估计标签噪声并量化数据质量。

这种与模型无关的方法使开发者能够诊断和纠正数据集错误，然后重新训练模型以提高健壮性和准确性。我们的团队在生产环境中成功使用了 Cleanlab，确认了它在实际环境中的有效性。在 AI 工程项目中，我们推荐它作为促进数据标准化和改善数据集质量的有价值的工具。

65. Context7

试验

Context7 是一个 MCP 服务器，旨在解决 AI 生成代码中的不准确性问题。鉴于 LLM 依赖于过时的训练数据，Context7 通过直接从框架的源代码库中提取最新的文档和可运行的代码示例，并在提示时将其注入到 LLM 的上下文窗口中，从而确保 AI 生成的代码对于项目所使用的库和框架而言是准确、最新且版本特定的。根据我们的经验，Context7 极大地减少了代码幻觉和对过时训练数据的依赖。您可以将其配置到 Claude Code、Cursor 或 VS Code 等 AI 代码编辑器中，用于生成、重构或调试依赖于特定框架的代码。

66. 数据契约 CLI

试验

数据契约 CLI 是一个专为使用数据契约规范而设计开源命令行工具。它帮助你创建和编辑数据契约，更重要的是，它让你能够根据契约验证数据，这对于确保数据产品的完整性和质量至关重要。

该 CLI 为多种模式定义（Avro、SQL DDL、开放数据契约标准等）提供广泛支持，并且可以比较不同的契约版本，从而立即检测到重大变更。我们发现它在数据网格领域特别有用，可以通过 CI/CD 集成在数据产品之间实现契约治理。这种方法减少了人为错误，并确保跨服务数据交换中的数据质量、完整性和兼容性。

67. Databricks Assistant

试验

Databricks Assistant 是一款直接集成于 Databricks 平台的 AI 驱动对话式工具，为数据专业人员充当上下文相关的结对编程助手。与通用代码助手不同，Databricks Assistant 能原生理解 Databricks 环境和数据上下文，包括来自 Unity Catalog 的元数据。其能力不仅限于生成代码片段，还能编写复杂的多步 SQL 和 Python 查询，诊断错误，并提供详尽且针对工作空间的解释说明。对于已选用 Databricks 生态的组织，该助手有助于提升生产效率，降低复杂数据任务的入门门槛。

68. Hoppscotch

试验

Hoppscotch 是一个轻量级开源工具，用于 API 开发、调试、测试和共享。它支持多种协议，包括 HTTP、GraphQL 和 WebSocket，并提供适用于 Web、桌面和 CLI 环境的跨平台客户端。

虽然 API 工具领域竞争激烈，有 Postman、Insomnia 和 Bruno 等替代方案，Hoppscotch 以其轻量体积和注重隐私的设计脱颖而出。它不收集分析数据，采用本地优先存储，并支持自托管。对于希望以直观方式共享 API 脚本，同时保持数据隐私的组织来说，这是一个很好的选择。

69. NVIDIA DCGM Exporter

试验

NVIDIA DCGM Exporter 是一个开源工具，帮助团队在大规模分布式 GPU 训练中进行监控。它将 NVIDIA 数据中心 GPU 管理器（DCGM）的专有遥测数据转换为兼容标准监控系统的开放格式。Exporter 暴露关键的实时指标——包括 GPU 利用率、温度、功耗和 ECC 错误计数——涵盖 GPU 和主机服务器。这种可见性对于微调自定义 LLM 或运行长时间、高强度 GPU 训练作业的组织至关重要。滞后效应——即单个缓慢工作节点限制整个流程——可能降低吞吐量超过 10%，并浪费多达 45% 的分配 GPU 小时。DCGM Exporter 为云原生大规模环境设计，可与 Prometheus 和 Grafana 无缝集成，帮助确保每个 GPU 在最佳性能范围内运行。

70. RelationalAI

试验

当大量多样化的数据被引入 Snowflake 时，其中固有的关系和隐含规则可能会被掩盖。作为一款 Snowflake 原生应用，RelationalAI 使团队能够构建复杂模型，以捕捉有意义的概念、定义核心业务实体，并将复杂逻辑直接嵌入到 Snowflake 表之上。其强大的图推理器允许用户基于这些模型创建、分析和可视化关系知识图谱。内置的算法有助于探索图结构并揭示隐藏模式。对于管理海量、快速变化数据集的组织而言，构建知识图谱对于主动监控以及生成更丰富、更可操作的洞见至关重要。

71. UX Pilot

试验

UX Pilot 是一款支持多阶段 UX 设计流程的 AI 工具——从线框图，到高保真视觉设计，以及设计评审。它可接受文本或图像输入，并能够自动生成界面、流程和布局。其 Autoflow 功能可创建用户流程过渡，而 Deep Design 则生成更丰富、更细致的输出。UX Pilot 还提供 Figma 插件，可将生成的设计导出以在标准设计工具中进行精细化处理。 我们的团队曾使用 UX Pilot 进行创意和灵感探索，在 Crazy 8's 练习中生成多种方案，并将项目故事列表转化为产品愿景板和 Epic 级别的设计概念。像 UX Pilot 这样的工具还使非设计人员（如产品经理）能够 快速创建原型 并收集早期利益相关者反馈，这在 AI 辅助设计工作流中正成为一种日益增长的趋势。

72. v0

试验

v0 自上次在 Radar 中亮相以来已有所发展。它现在包含设计模式，进一步降低了产品经理创建和调整自助式 UI 原型的门槛。最新版本引入了内部模型，具有大上下文窗口和多模态能力，使 v0 能够从文本和视觉输入生成并改进 UI。另一个值得注意的新增功能是其代理模式，允许系统将更复杂的工作拆解，并为每个任务选择合适的模型。然而，这项功能仍处于早期阶段，目前收到的反馈褒贬不一。

73. Augment Code

评估

Augment Code 是一款 AI 编码助手，可在大型代码库中提供深度、具备上下文感知能力的开发支持。它的亮点在于先进的上下文工程，即使在代码频繁变更时，也能实现快速的代码索引更新与高效检索。Augment 支持包括 Claude Sonnet 4 和 4.5、GPT-5 在内的大型模型，可集成 GitHub、Jira、Confluence，并支持用于外部工具互操作性的 模型上下文协议（MCP）。它为复杂的代码库变更（如重构、依赖升级、架构更新等）提供逐步指导，并具备能反映项目特定依赖关系的个性化内联补全功能。Augment 还支持团队直接在 Slack 内查询和共享代码洞察，促进协作。

74. Azure AI 文档智能

评估

Azure AI 文档智能（前称 Form Recognizer）可以从非结构化文档中提取文本、表格和键值对，并将其转换为结构化数据。它使用预训练的深度学习模型来理解布局和语义，并且可以通过无代码界面训练自定义模型以处理特定格式。不过，在某些情况下，高级用户可能需要自定义的微调界面。 我们的一个团队报告称，ADI 显著减少了人工数据录入，提高了数据准确性并加快了报告速度，从而推动了更快速的数据驱动决策。与 Amazon Textract 和 Google Document AI 类似，它提供企业级文档处理能力，并具有强大的布局理解能力。一种新兴的开源替代方案是 IBM 的 Docling，它提供更灵活、以代码为中心的结构化数据提取方法。与传统 OCR 工具相比，ADI 不仅捕获文本，还捕获结构和关系，使其易于集成到下游数据管道中。不过，我们也观察到在同步用户工作流中嵌入时偶尔会有延迟，因此建议主要用于异步处理。

75. Docling

评估

Docling 是一个开源的 Python 和 TypeScript 库，用于对非结构化数据进行高级文档处理。它解决了常被忽视的“最后一公里”问题，即将真实世界的文档——如 PDF 和 PowerPoint——转换为干净、可机器读取的格式。与传统提取器不同，Docling 使用基于计算机视觉的方法来解析文档的布局和语义结构，使其输出对于增强检索生成（RAG）流水线特别有价值。它可将复杂文档转换为结构化格式，如 JSON 或 Markdown，并支持 LLM 的结构化输出 等技术。这与 ColPali 不同，后者直接将页面图像输入视觉 - 语言模型以进行检索。

Docling 的开源特性和基于 Python 的核心（建立在自定义的 Pydantic 数据模型上）为团队提供了灵活的自托管替代方案，相比于 Azure 文档智能、Amazon Textract 和 Google Document AI 等专有云工具更具自主性。该项目由 IBM Research 支持，开发快速，并提供可即插即用的架构，可与 LangGraph 等其他框架集成，非常值得构建生产级 AI 数据管道的团队进行评估。

76. E2B

评估

E2B 是一个开源工具，可在云端的安全隔离沙箱环境中运行 AI 自动生成的代码。这些沙箱基于 Firecracker 微虚拟机构建，支持智能体安全地执行代码、分析数据、开展研究或操作虚拟机。这样，你可以构建与部署企业级 AI 智能体，并完全控制和保障其运行环境的安全性。

77. Helix 编辑器

评估

近年来，简单文本编辑器有所回潮，试图取代命令行中的热门选择 Vim。Helix 就是其中之一，在拥挤的市场中与 Neovim 以及最近的 Kakoune 竞争。Helix 自称为“后现代文本编辑器”，略带玩笑意味，具备多光标、Tree-sitter 支持以及集成的语言服务器协议（LSP）支持，这也是最初吸引我们注意的地方。Helix 正在积极开发中，并计划引入插件系统。总体而言，它是一款轻量级的模态编辑器，对 Vim 用户来说感觉熟悉，同时增加了一些现代化便利功能。

78. Kueue

评估

Kueue 是一个 Kubernetes 原生的作业队列控制器，用于管理配额和资源消耗。它提供了处理不同优先级和资源需求的 Kubernetes 工作负载的 API，充当作业级管理器，决定何时允许或驱逐作业。Kueue 旨在实现高效的资源管理、作业优先级排序和高级调度，帮助优化 Kubernetes 环境中的工作负载执行，尤其适用于使用 Kubeflow 等工具的机器学习工作负载。它与 cluster-autoscaler 和 kube-scheduler 协同工作，而非替代它们，重点根据作业顺序、配额、优先级和拓扑感知进行作业准入。作为 Kubernetes 特别兴趣组（SIG）生态系统的一部分，Kueue 遵循其开发标准。

79. MCP-Scan

评估

MCP-Scan 是一种为 Model Context Protocol (MCP) 服务器设计的安全扫描器，拥有扫描和代理两种运行模式。在扫描模式下，它分析配置和工具描述以检测已知漏洞，例如提示词注入、工具投毒和有害流分析。在代理模式下，MCP-Scan 作为智能体系统与 MCP 服务器之间的桥梁，持续监控运行时流量，并可强制执行自定义安全规则和防护措施，包括工具调用校验、敏感信息检测和数据流约束。该工具为智能体系统提供了主动型安全防线，即使恶意提示被接受，也能确保智能体无法执行有害操作。MCP-Scan 是为新兴的智能体系统安全领域量身打造的解决方案。

80. oRPC

评估

oRPC (OpenAPI Remote Procedure Call) 在 TypeScript 中提供端到端的类型安全 API，同时完全遵循 OpenAPI 规范。它可以自动生成完整的 OpenAPI 规范，简化集成和文档。我们发现 oRPC 特别适用于集成。虽然 tRPC 和 ElysiaJS 等替代方案通常需要采用新的框架来获得类型安全性，但 oRPC 可以无缝集成现有的 Node.js 框架，包括 Express, Fastify, Hono 和 Next.js。这种灵活性使其成为希望采用端到端类型安全性到现有 API 的团队的一个很好的选择，而无需破坏性的重构。

81. Power user for dbt

评估

Power user for dbt 是一个 Visual Studio Code 扩展，能够直接集成 dbt 和 dbt Cloud 环境。dbt 一直是我们最喜欢的工具之一，因此任何能够提升其易用性的工具都值得我们欢迎。此前，开发者需要依赖多种工具来验证 SQL 代码或在 IDE 外检查模型血缘关系。而使用该扩展后，这些功能已内置于 VS Code 中，提供代码自动补全、实时查询结果以及可视化的模型和列血缘关系。最后一项功能使在模型之间导航变得轻松。我们的团队报告称，该插件减少了流水线错误，并提升了整体开发体验。如果你使用 dbt，我们强烈建议试用这个工具。

82. Serena

评估

Serena 是一个强大的编程工具包，为像 Claude Code 这样的编程智能体提供类 IDE 功能，用于语义化代码检索和编辑。通过在符号层面操作并理解代码的关系结构，Serena 大幅提升了 token 的使用效率。编程智能体无需读取整个文件或依赖粗糙的字符串替换，而可以使用精准的 Serena 工具，例如 find_symbol、find_referencing_symbols 和 insert_after_symbol 来定位和编辑代码。虽然在小型项目中效果有限，但随着代码库增长，这种效率提升极具价值。

83. SweetPad

评估

SweetPad 扩展让开发者能够在 Apple 平台使用 VS Code 或 Cursor 完成整个 Swift 应用的开发生命周期。它通过集成 xcodebuild、xcode-build-server 和 swift-format 等核心工具，免去了频繁切换到 Xcode 的需求。开发者现在可以直接在 IDE 内构建、运行、调试 iOS、macOS 和 watchOS 的 Swift 应用，同时还可管理模拟器并在无需打开 Xcode 的情况下部署到设备。

84. Tape/Z（用于 Z/OS 的汇编程序分析工具集）

评估

Tape/Z（用于 Z/OS 的汇编程序分析工具集）是一个专为主机 HLASM（高级汇编）代码分析而开发的工具集，由 Thoughtworks 工程师创建。它具备解析、控制流图构建、依赖追踪和流程图可视化等功能。长期以来，主机领域一直缺乏开源、社区驱动的工具，大多数可选方案都属于专有产品或绑定于特定厂商生态。Tape/Z 通过提供易于访问、可脚本化的分析能力，正在弥补这一空白。与之配套的还有 COBOL REKT——我们多次在客户项目中使用的一款 COBOL 工具，这两者标志着主机系统迈向现代化、开发者友好工具的积极进展。

语言和框架

采纳

85. Fastify
86. LangGraph
87. vLLM

试验

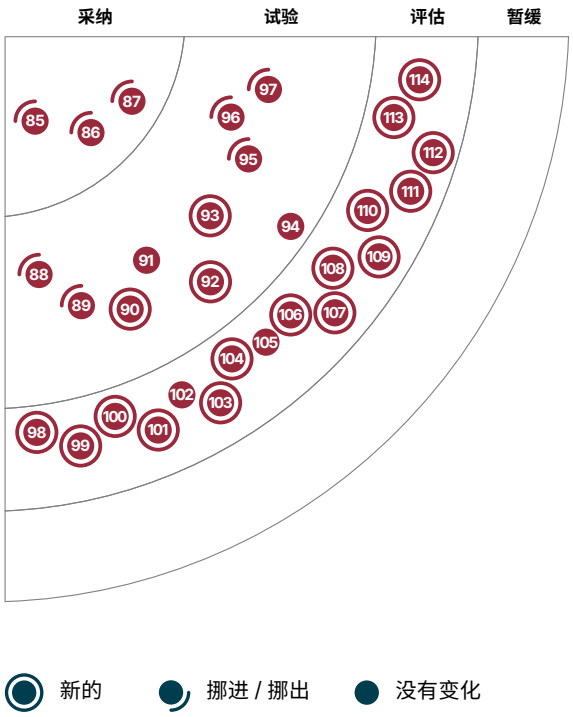
88. Crossplane
89. DeepEval
90. FastMCP
91. LiteLLM
92. MLForecast
93. Nuxt
94. Phoenix
95. Presidio
96. Pydantic AI
97. Tauri

评估

98. Agent Development Kit（ADK）
99. Agno
100. assistant-ui
101. AutoRound
102. Browser Use
103. DeepSpeed
104. Drizzle
105. Java 后量子密码学
106. kagent
107. LangExtract
108. Langflow
109. LMCache
110. Mem0
111. 开放安全控制评估语言 (OSCAL)
112. OpenInference
113. Valibot
114. Vercel AI SDK

暂缓

—



85. Fastify

采纳

我们对 Fastify 的体验持续保持积极评价。它是一款快速、灵活、低开销的 Node.js Web 框架。它提供了极简 Web 框架所需的所有核心功能，包括解析、验证和序列化，同时拥有强大的插件系统和活跃的社区支持。我们的团队在使用 Fastify 时，相比 Express.js 等替代方案，并未发现明显的缺点，反而获得了可衡量的性能提升，使其成为 Node.js 极简 Web 开发中一个引人注目的选择。

86. LangGraph

采纳

LangGraph 是一个用于使用 LLM 构建有状态的多智能体应用的编排框架。它提供节点和边等底层原语，以及内置功能，使开发人员能够精细控制智能体的工作流、内存管理和状态持久化。这意味着开发人员可以从一个简单的预构建图入手，并扩展到复杂且不断演进的智能体架构。通过支持流式处理、高级上下文管理以及模型回退和工具错误处理等弹性模式，LangGraph 使你能够构建健壮的生产级智能体应用。其基于图的方法确保了可预测的、可定制的工作流，并简化了调试和扩展。我们的团队使用 LangGraph 构建多智能体系统取得了很好的效果，这得益于其轻量级和模块化的设计。

87. vLLM

采纳

vLLM 是一个高吞吐量、内存高效的 LLM 推理引擎，可在云端或本地运行。它支持多种模型架构和流行的开源模型。我们的团队在 GPU 平台（如 NVIDIA DGX 和 Intel HPC）上部署了容器化的 vLLM 工作节点，托管包括 Llama 3.1 (8B 和 70B)、Mistral 7B 和 Llama-SQL 等模型，用于开发者编码辅助、知识搜索和自然语言数据库交互。vLLM 与 OpenAI SDK 标准兼容，实现一致的模型服务。Azure 的 AI Model Catalog 使用基于 vLLM 的自定义推理容器以提升服务性能，并将 vLLM 作为默认推理引擎，因其高吞吐量和高效内存管理而被广泛采用。vLLM 框架已成为大规模模型部署的首选方案。

88. Crossplane

试验

自上次在技术雷达中亮相以来，Crossplane 的采用率持续增长，尤其是在扩展 Kubernetes 集群方面。在我们的实践中发现，Crossplane 在特定用例中表现出色，而非作为通用的基础设施即代码（IaC）工具。我们早先的观察仍然成立：Crossplane 最适合作为 Kubernetes 内部署工作负载的辅助工具，而不是完全替代 Terraform 等工具。那些将 Crossplane 作为主要 IaC 解决方案的团队往往举步维艰，而那些务实地将其用于特定定制用例的团队则取得了显著成效。将资源生命周期管理下放给 Crossplane，同时利用 XRD API 进行轻量级定制，证明尤其有效。当管理那些生命周期简单但并非 Kubernetes 原生支持的资源时，Crossplane 的价值尤为突出。虽然它现在可以创建 Kubernetes 集群——这一功能以前缺失——我们仍建议在将 Crossplane 作为 Terraform 完全替代方案时保持谨慎。根据我们的经验，最佳做法是以 IaC 作为基础层，再在其上使用 Crossplane 来满足特定需求。

89. DeepEval

试验

DeepEval 是一个开源的、基于 Python 的 LLM 性能评估框架。它可以用于评估 检索增强生成 (RAG) 和其他使用框架（如 LlamaIndex 或 LangChain）构建的应用程序，也可以用于基线和基准模型。DeepEval 超越了单词匹配得分，评估准确性、相关性和一致性，在现实场景中提供更可靠的评估。它包括诸如幻觉检测、答案相关性和超参数优化等指标，并支持 GEval 创建自定义、用例特定的指标。我们的团队使用 DeepEval 使用 LLM 作为评判者技术微调智能体输出。它与 pytest 和 CI/CD 管道集成，使其易于采用并具有持续评估的价值。对于在受监管环境中构建 LLM 应用程序的团队，由英国人工智能安全研究所开发的 Inspect AI 提供了一个替代方案，更专注于审计和合规性。

90. FastMCP

试验

Model Context Protocol (MCP) 正快速成为为 LLM 应用提供上下文和工具的标准。然而，实施 MCP 服务器通常涉及大量的设置、协议处理和错误管理样板代码。FastMCP 是一个 Python 框架，通过抽象协议复杂性并允许开发者使用直观的 Python 装饰器定义 MCP 资源和工具，从而简化了这一过程。这种抽象使团队能够专注于业务逻辑，从而实现更清晰、更易维护的 MCP 实现。

虽然 FastMCP 1.0 已经集成到官方 SDK 中，但 MCP 标准仍在快速发展。我们建议关注 2.0 版本发布，并确保团队与官方规范的更新保持同步。

91. LiteLLM

试验

LiteLLM 是一个 SDK，可通过标准化的 OpenAI API 格式 实现与多个 LLM 服务商的无缝集成。它支持多种服务商和模型，为文本生成、嵌入和图像生成提供统一接口。通过屏蔽不同服务商的 API 差异，LiteLLM 简化了集成流程，并会自动将请求路由至对应模型端点。其代理框架还包含生产级特性，如安全控制、缓存、日志记录、限流和负载均衡。随着企业将 AI 应用更深入地嵌入工作流，治理和可观测性变得尤为重要。我们的团队一直将 LiteLLM 用作 AI 网关，以实现企业级 AI 使用的标准化、安全管控和可见性。

92. MLForecast

试验

MLForecast 是一个用于时间序列预测的 Python 框架和库，它将机器学习模型应用于大规模数据集。它简化了通常复杂的自动特征工程过程——包括滞后特征、滑动统计和基于日期的特征——并且是少数原生支持分布式计算框架（如 Spark 和 Dask）的库之一，从而保证了可扩展性。它还支持使用符合预测等方法进行概率预测，为预测不确定性提供量化指标。在我们的评估中，MLForecast 能够高效处理数百万数据点，并持续优于同类工具。对于希望在高数据量场景下快速实现时间序列预测的团队，MLForecast 是一个非常有吸引力的选择。

93. Nuxt

试验

Nuxt 是一个基于 Vue.js 构建的有观点的元框架，用于创建全栈 Web 应用，常被称为 “Vue.js 版的 Next.js”。与其 React 对应框架类似，Nuxt 提供 SEO 友好的功能，如预渲染、服务器端渲染（SSR）和元数据管理，使其成为在 Vue.js 技术栈上构建性能导向、SEO 优化网站的强力选择。

Nuxt 由 Vercel 支持（同样公司也开发了 Next.js），并拥有强大的社区和官方及第三方模块生态。这些模块简化了图像处理、站点地图生成和 Tailwind CSS 等功能的集成。对于希望使用 Vue.js 构建 SEO 友好应用的团队来说，Nuxt 是一个全面、有观点的框架选择。

94. Phoenix

试验

我们依然在使用 Phoenix（一个由 Elixir 编写的服务端 Web MVC 框架）中获得了良好的体验。Phoenix 在借鉴 Ruby on Rails 的快速开发和开发者体验的同时，也进一步推进了函数式编程范式的发展。

本期我们重点关注 Phoenix LiveView 1.0 的发布。LiveView 是一种基于 HTML-over-the-wire 的解决方案——类似于 HTMX 或 Hotwire——它使开发者能够完全使用服务端渲染的 HTML 构建丰富的实时用户体验。与通常仅通过 HTML 局部更新的类似技术不同，LiveView 通过 LiveComponents 提供了完整的组件化架构，支持组件组合、属性传递、状态管理和生命周期钩子等特性，与 React 或 Vue.js 类似。LiveView 将 Phoenix 的高生产力与可扩展性结合起来，同时具备强大的复杂性管理能力，非常适合在无需完整 JavaScript 框架的情况下构建高度交互的前端应用。

95. Presidio

试验

Presidio 是一个数据保护 SDK，用于在结构化和非结构化文本中 识别 和 匿名化 敏感数据。它利用命名实体识别、正则表达式和基于规则的逻辑，检测个人身份信息（PII），例如信用卡号、姓名和位置。Presidio 支持 自定义 实体识别器和去标识化流水线，使组织能够根据其隐私和合规需求进行定制。我们的团队已在具有严格数据共享控制的企业环境中，将 Presidio 用于与 LLM 集成。尽管它实现了敏感信息检测的自动化，但它并非万无一失，可能会遗漏或错误识别实体。团队在依赖其结果时，请务必保持谨慎。

96. Pydantic AI

试验

Pydantic AI 持续证明自己是一个稳定、支持良好的开源框架，适用于在生产环境中构建生成式 AI 智能体。它基于可靠的 Pydantic 基础，提供强类型安全、通过 OpenTelemetry 实现的一流可观测性，以及内置评估工具。2025 年 9 月 4 日发布的 1.0 版本标志着其成熟度的重要里程碑。自那以后，我们发现它因简单性和可维护性而可靠且广泛采用，加入了类似 LangChain 和 LangGraph 等其他流行智能体框架的行列。

近期更新使实现模型上下文协议（MCP）服务器和客户端更为便捷，并增加了对 AG-UI 和 A2A 等新兴标准的支持。凭借其清晰的 API 和不断增长的生态系统，Pydantic AI 已成为我们团队在 Python 中构建生产就绪生成式 AI 应用的有力选择。

97. Tauri

试验

Tauri 是一个使用单一 Web UI 代码库构建高性能桌面应用的框架。与传统的 Web 包装器如 Electron 不同，Tauri 基于 Rust 构建，并利用操作系统的原生 Webview，从而生成更小的二进制文件并提升安全性。我们在几年前首次评估了 Tauri；此后，它已扩展到支持 iOS 和 Android。最新版本引入了更灵活的权限与作用域模型，替换了旧的权限列表，并强化了进程间通信（IPC）层，支持原始数据传输并提升性能。这些更新均经过外部安全审计。结合各大应用商店的官方分发指南，这些改进进一步巩固了 Tauri 在跨平台开发领域的地位。

98. Agent Development Kit（ADK）

评估

Agent Development Kit（简称 ADK）是一个用于开发和部署 AI 智能体的框架，它将现代软件工程方法引入智能体构建，而不仅仅依赖提示。ADK 引入了熟悉的抽象概念，如类、方法、工作流模式和命令行界面（CLI）支持。与 LangGraph 或 CrewAI 等框架相比，ADK 的优势在于其与 Google AI 基础设施的深度集成，提供企业级的上下文绑定、数据访问和监控能力。ADK 还注重互操作性，支持工具封装器以及用于智能体间通信的 A2A 协议。对于已在 GCP 上投入的组织而言，ADK 为构建可扩展、安全且易于管理的智能体架构提供了有前景的基础。尽管它仍处于早期发展阶段，但这表明了 Google 正在迈向原生的全栈智能体开发环境。我们建议密切关注其成熟度和生态系统的发展。

99. Agno

评估

Agno 是一个用于构建、运行和管理多智能体系统的框架。它提供了创建完全自主智能体或受控的、基于步骤的工作流的灵活性，并内置了对人机协作（Human-in-the-Loop）、会话管理、记忆和知识的支持。我们赞赏其对效率的关注，它实现了令人印象深刻的智能体启动速度和极低的内存消耗。Agno 还附带了其运行时 AgentOS，这是一个基于 FastAPI 的应用程序，带有集成的控制平面，可用于简化的智能体系统测试、监控和管理。

100. assistant-ui

评估

assistant-ui 是一个用于 AI 对话界面的开源 TypeScript 和 React 库。它处理了聊天 UI 实现中的复杂环节——比如消息流式处理、消息编辑与分支切换的状态管理，以及常见的用户体验要素——同时允许开发者利用 Radix primitives 自行设计组件。它支持与主流运行环境集成，包括 Vercel AI SDK 和 LangGraph，并为复杂应用场景提供可自定义的运行时方案。我们已经成功基于 assistant-ui 构建了简单的聊天界面，并对其效果感到满意。

101. AutoRound

评估

英特尔的 AutoRound 是一种先进的量化算法，能够在几乎不损失精度的前提下压缩大型 AI 模型（如 LLM 和图文多模态模型）。它采用符号梯度下降优化，以及跨不同层的混合比特宽度方案，可将模型精度降至极低的 2–4 比特范围，从而实现最佳效率。AutoRound 的量化过程也非常迅速：在单块 GPU 上，仅需数分钟即可完成对 70 亿参数模型的量化。由于 AutoRound 能与 vLLM 和 Transformers 等主流推理引擎集成，所以成为模型量化的有吸引力方案。

102. Browser Use

评估

Browser Use 是一个开源的 Python 库，使基于大型语言模型（LLM）的智能体能够操作网页浏览器并与 Web 应用程序交互。它可以执行导航、输入数据、提取文本以及管理多个标签页，以便在不同应用之间协调操作。当 AI 智能体需要访问、处理或检索网页内容时，该库尤其有用。它支持多种 LLM，并基于 Playwright，结合视觉理解与 HTML 结构提取，实现更丰富的网页交互。我们的团队将 Browser Use 与 Pytest 框架和 Allure 报告系统集成，用于探索基于 LLM 的自动化测试。测试步骤以自然语言编写，由智能体执行，并在断言或失败时自动捕获截图。其目标是在开发完成后，通过自动从 Confluence 拉取测试用例，实现非工作时段的 QA 测试。早期结果较为理想，但智能体在任务结束后的响应中往往缺乏详细的错误描述，因此需要自定义错误报告机制来补充。

103. DeepSpeed

评估

DeepSpeed 是一个 Python 库，用于优化分布式深度学习的训练和推理。对于训练，它集成了 Zero Redundancy Optimizer (ZeRO) 和 3D 并行等技术，以高效地在数千 GPU 上扩展模型。对于推理，它结合了张量并行、流水线并行、专家并行和 ZeRO 并行，并通过自定义内核和通信优化来最小化延迟。DeepSpeed 支持世界上一些最大的语言模型，包括 Megatron-Turing NLG（530B）和 BLOOM（176B）。它兼容稠密模型和稀疏模型，提供高系统吞吐量，并允许在多 GPU 资源受限的环境下进行训练或推理。该库可与流行的 Hugging Face Transformers、PyTorch Lightning 和 Accelerate 无缝集成，是大规模或资源受限深度学习工作负载的高效解决方案。

104. Drizzle

评估

Drizzle 是一个轻量级的 TypeScript ORM。与 Prisma ORM 不同，它为开发者提供了既简单的类 SQL API，也更传统的 ORM 风格查询接口。它还支持从现有数据库提取模式，从而同时支持数据库优先和代码优先的方法。Drizzle 针对无服务器环境设计：它具有小巧的包体积，并支持预编译语句，使 SQL 查询可以预先编译，从而让数据库驱动直接执行二进制 SQL，而无需每次解析查询。其简洁性和对无服务器环境的支持，使 Drizzle 成为 TypeScript 生态中 ORM 的理想选择。

105. Java 后量子密码学

评估

量子计算机正快速发展，SaaS 服务如 AWS Braket 已经能够在多种架构上提供量子算法的访问能力。自今年三月起，Java 24 引入了 Java 后量子密码学，增加了对后量子加密算法的支持，例如 ML-KEM 和 ML-DSA；同时，.Net 10 也扩展了其支持。我们的建议很简单: 如果你在这些语言中构建软件，请立即开始采用量子安全算法，以保障系统的未来适应性。

106. kagent

评估

Kagent 是一个在 Kubernetes 集群中运行智能体的开源框架。它使基于大语言模型（LLM）的智能体能够通过 Kubernetes 原生 API 和 Model Context Protocol（MCP）集成，规划并执行运维任务，例如诊断问题、修复配置或与可观测性工具交互。其目标是将“AgentOps”引入云原生基础设施，将声明式管理与自主推理相结合。 作为一个 CNCF Sandbox 项目，Kagent 的引入需要谨慎，特别是在赋予 LLM 运维管理能力时所带来的潜在风险。诸如 AI 有害流分析（toxic flow analysis）等技术在评估和缓解这些风险时尤其有价值。

107. LangExtract

评估

LangExtract 是一个可根据用户定义的指令，使用大型语言模型（LLM）从非结构化文本中提取结构化信息的 Python 库。它可以处理领域特定的材料——例如临床记录和报告——并在识别和组织关键信息的同时，让每个提取的数据点都能追溯到其来源。提取的实体可导出为 .jsonl 文件，这是一种语言模型数据的标准格式，并可通过交互式 HTML 界面进行可视化，以便进行上下文审查。我们的团队评估了 LangExtract 在实体提取以填充领域知识图谱方面的能力，发现它在将复杂文档转化为结构化、机器可读的格式方面卓有成效。

108. Langflow

评估

Langflow 是一个用于构建和可视化 LLM 工作流的开源低代码平台。它基于 LangChain 构建，并允许开发者通过拖拽界面将提示词、工具、向量数据库和内存组件链接起来，同时仍支持自定义 Python 代码来实现高级逻辑。它特别适用于在不编写完整后端代码的情况下对智能体应用进行原型设计。然而，Langflow 仍然相对较新，在生产使用方面存在一些小缺点。我们通常对低代码平台的谨慎态度在这里同样适用。话虽如此，我们喜欢 Langflow 的工作流可以定义为代码并进行版本控制，这可以缓解低代码平台的一些缺点。

109. LMCache

评估

LMCache 是一个用于加速大语言模型（LLM）服务基础设施的键值对（KV）缓存解决方案。它作为跨 LLM 推理引擎池的专门缓存层，存储可能被多次处理的文本的预计算 KV 缓存条目，如聊天历史或文档集合。通过将这些值持久化到磁盘上，预填充计算可以从 GPU 卸载，减少首令牌时间（TTFT）并降低 RAG 管道、多轮聊天应用和智能体系统等高要求工作负载的推理成本。你可以将 LMCache 与主要推理服务器（如 vLLM 或 NVIDIA Dynamo 集成，并认为值得评估其对你的设置的影响。

110. Mem0

评估

Mem0 是一个专为 AI 智能体设计的内存层。初级的记忆方法通常将整个聊天历史存储在数据库中，并在未来的对话中复用，这会导致过度的令牌使用。Mem0 用一个更复杂的架构替代了这种方法，该架构将内存分为短期回忆和智能长期记忆层，后者只提取和存储重要的事实和关系。其架构结合了用于语义相似性的向量存储和用于理解时间和关系数据的知识图谱。这种设计显著减少了上下文令牌的使用，同时使智能体能够保持长期认知，这对个性化和许多其他场景极其有用。

111. 开放安全控制评估语言(OSCAL)

评估

开放安全控制评估语言（OSCAL）是一种开放、机器可读的信息交换格式，旨在提升合规和风险管理的自动化水平，帮助团队摆脱基于文本的手工处理方式。在美国国家标准与技术研究院（NIST）的领导下，OSCAL 提供了基于 XML、JSON 和 YAML 的标准化表达方式，用于描述与行业框架（如 SOC 2 和 PCI）以及政府框架（如美国的 FedRAMP、新加坡的网络安全控制目录和澳大利亚的 信息安全手册）相关的安全控制。

尽管 OSCAL 目前在公共部门之外尚未被广泛采用，其生态系统也仍在不断完善，但我们对其带来的潜力感到兴奋。OSCAL 有望简化安全评估流程，减少对电子表格以及走形式的“打勾”操作的依赖，甚至在集成进合规即代码（compliance-as-code）和持续合规平台后，实现合规自动化。

112. OpenInference

评估

OpenInference 是一套约定和插件集，作为 OpenTelemetry 的补充，专为观测 AI 应用而设计。它为机器学习框架和 类库 提供了标准化的工具，帮助开发者追踪 LLM 的调用，以及向量存储检索、外部工具（如 API 和搜索引擎）调用等周围上下文。Span 数据可以导出到任何兼容 OTEL 的收集器，确保与现有遥测流水线保持一致。我们之前曾提及 Langfuse，这是一个常用的 LLM 可观测性平台—— OpenInference SDK 可以将追踪数据记录到 Langfuse 以及其他兼容 OpenTelemetry 的可观测性平台中。

113. Valibot

评估

Valibot 是一个 TypeScript 的模式验证库。与其他流行的 TypeScript 验证库如 Zod 和 Ajv 类似，它提供类型推断，但其模块化设计使其与众不同。这种架构允许打包工具进行高效的 tree shaking 和代码拆分，仅包含实际使用的验证函数。在最佳场景下，Valibot 可将包体积相比 Zod 减少多达 95%。对于对包大小敏感的环境，如客户端验证或无服务器函数，Valibot 是一个颇具吸引力的选择。

114. Vercel AI SDK

评估

Vercel AI SDK 是一个开源的全栈工具包，用于在 TypeScript 生态系统中构建由 AI 驱动的应用和智能体。它包含两个主要组件: AI SDK Core 用于标准化与模型无关的 LLM 调用，支持文本生成、结构化对象生成以及工具调用；AI SDK UI 则简化了前端开发，提供流式响应、状态管理和在 React、Vue、Next.js、Svelte 中的实时界面更新，类似于 assistant-ui。对于已经使用 TypeScript 和 Next.js 的团队而言，Vercel AI SDK 提供了一种快速、无缝的方式来构建具有丰富客户端体验的 AI 应用程序。

想要了解技术雷达最新的新闻和洞见?

点击订阅，以接收每两个月一次、来自 Thoughtworks 的技术洞察和未来趋势探索邮件。

现在订阅

Thoughtworks 是一家全球性软件及技术咨询公司，集战略、设计和工程技术咨询服务于一体，致力于推动数字创新。我们在 19 个国家 / 地区的 48 个办公室拥有超过 10,500 名员工。 在过去的 30 年里，我们为全球各地的众多合作伙伴倾力服务，与客户一起创造了非凡的影响力，帮助他们以技术为优势解决复杂的业务问题。

/thoughtworks

Design. Engineering. AI.